Preparing depth imaging applications for Exascale challenges and impacts

Asma Farjallah

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Université de Versailles Saint-Quentin-en-Yvelines | École Doctorale des Sciences et Technologies de Versailles

Preparing Depth Imaging Applications for Exascale Challenges and Impacts

---

Étude de l'adéquation des machines Exascale pour les algorithmes implémentant la méthode du Reverse Time Migration

THÈSE

présentée et soutenue publiquement le 16 Décembre 2014

pour l'obtention du

Doctorat de l'université de Versailles Saint-Quentin-en-Yvelines

(spécialité informatique)

par

Asma Farjallah

Composition du jury

Directeur de thèse :	William Jalby	- Professeur, Université de Versailles
Président :	Nahid Emad	- Professeur, Université de Versailles
Rapporteurs :	François Bodin	- Professeur, Université de Rennes 1
	Jean-Luc Lamotte	- Professeur, Université Pierre & Marie Curie
Examinateurs :	Henri Calandra	- Expert, Total E&P, USA
	Philippe Thierry	- Docteur, Intel, Paris

Exascale computing research

Remerciements

Je tiens à remercier mon directeur de thèse, William Jalby, pour son accueil au laboratoire Exascale Computing Research et son aide tout au long de ma thèse. Je remercie également Henri Calandra pour m'avoir fait confiance et pour ses précieux conseils.

Je remercie du fond du cœur mes collègues et mes copains qui m'ont épaulé pendant ses quatre années. Eric et Sylvain, merci d'être toujours présents dans les pires et les meilleurs moments. Thomas et Marc, merci pour tout ce que vous m'avez appris et pour les longues discussions au tableau. Omar et Othman, merci pour vos encouragements et votre soutien pendant la préparation de la soutenance. Philippe, merci pour ta patience et tes conseils qui ne tarissent jamais. Issam, merci de toujours penser à moi.

A ma famille, j'offre cette thèse. Mon papa et ma maman, j'espère que cette thèse est à la hauteur de vos sacrifices et vos efforts sans lesquels je ne serais pas devenue celle que je suis. Je pense très fort à mes sœurs Emna et Eya et leur souhaite la réussite et le bonheur.

A l'élu de mon cœur, Wassim, j'offre cette thèse aussi. Merci d'être toujours là pour moi, pour m'encourager et m'aider. J'espère que la vie nous réserve que le bonheur et la joie de vivre.

À mes parents et Wassim.

Abstract

As we are expecting Exascale systems for the 2018-2020 time frame, performance analysis and characterization of applications for new processor architectures and large scale systems are important tasks that permit to anticipate the required changes to efficiently exploit the future HPC systems. The objective is to assess the portability of the application and identify the key challenges for both the application and the architecture. This thesis focuses on seismic imaging applications used for modeling complex physical phenomena, in particular the depth imaging application called Reverse Time Migration (RTM). The study follows two main axes. The first one is FDTD, the computational core of RTM, the second one is the communication and IO of the full RTM application. A deep understanding of the interaction of these kernels with the underlying architecture is the key to predict the behavior of the overall application.

My first contribution consists in characterizing and modeling the performance of the FDTD kernel. I identify and explore the major tuning parameters influencing performance and the interaction between the architecture and the application.

The second contribution is an analysis to identify the challenges for a hybrid and heterogeneous implementation of FDTD for manycore architectures. We target Intel's first Xeon Phi co-processor, the Knights Corner. This architecture is an interesting proxy for our study since it contains some of the expected features of an Exascale system : concurrency and heterogeneity.

My third contribution is an extension of the performance analysis and modeling to the full RTM. This adds communications and IOs to the computation part. RTM is a data intensive application and requires the storage of intermediate values of the computational field resulting in expensive IO accesses.

My fourth contribution is the final measurement and model validation of my hybrid RTM implementation on a large system. This has been done on Stampede, a machine of the Texas Advanced Computing Center (TACC), which allow us to test the scalability up to 64 nodes each containing one 61-core Xeon Phi and two 8-core CPUs for a total close to 5000 heterogeneous cores.

The performance analysis and characterization study of RTM on a cluster hosting manycore architecture allow us to alleviate the hardware features that significantly impact the performance of the computational part. Furthermore, the modeling of the influence of the increasing concurrency and heterogeneity on the full RTM seismic imaging application is an important step to pave the way for the required code modernization and to pin-point the critical architectural bottlenecks

in the process for future Exascale systems co-design.

Keywords. Exascale systems, seismic imaging applications, co-design, modeling, performance characterization, Intel Knights Corner.

Contents

Introduction

Given the expected changes affecting the HPC systems and the constraints that may tailor their conception, application scientists need to lead off studies defining the main changes affecting the development of their future applications. They also need to define the new challenges they aim to overcome thanks to the increasing compute capability of the Exascale machines. These upstream studies also concern the current applications in order to be sure that their porting on Exaflopic systems is relevant.

This thesis is a feasibility study of porting seismic imaging applications as Reverse Time Migration (RTM) to the Exascale. Co-design is the approach we rely on to predict the behavior of these applications and anticipate solutions to remedy to the expected hardware and software challenges in Exascale systems. Algorithms and numerical schemes may consequently change in order to response to these challenges. My contributions are the following:

- **Performance modeling and characterization of FDTD.** We characterize the performance of finite-difference time-domain (FDTD) implementations widely used in seismic imaging applications like RTM since they represent their computational core. We consider two implementations. One in isotropic media as an introductory example and one in transverse anisotropic media tilted transverse isotropy (TTI) as it is commonly encountered in exploration campaigns. This study was made possible using analytical models, Roofline models and a sampling-based tool called Adaptive Sampling Kit (ASK).

- **Memory bandwidth cost modeling.** We demonstrate that the memory bandwidth has a great impact on the performance of the FDTD applications and we developed an analytical model predicting the DRAM traffic of these applications. This model can be used to predict performance on future Exascale systems.

- **FDTD on manycore architectures.** We then focus on the Intel Many-Integrated Core (MIC) architecture as it introduces new hardware features compared to Nehalem and Sandy Bridge. This highly parallel architecture is representative of the manycore technology trend in Exascale. We study the impact of concurrency using native implementations of the FDTD kernels. For this particular architecture, we highlight the optimizations performed in terms of data access and alignment, vectorization and cache blocking.

- **Exploring symmetric execution of RTM.** We also developed symmetric implementations in order to highlight the impact of heterogeneity on these kernels. We use two programming models, MPI and OpenMP and perform static load balancing in order to reduce communications overhead.
- **Performance analysis of RTM on large scale cluster.** We then focus on the multi-node implementation of the full application RTM. We develop analytical models that accounts for computation, communications and I/O as a prior study to porting on large scale systems. We make extrapolations based on these models for these systems and considering a simple velocity model a real velocity model. We also highlighted the bottlenecks, we encountered on Intel Xeon Phi nodes before the actual porting on a real system and we identified the impact of the small memory available on these co-processors and the need to over-decompose the compute grid which results in an overhead due to communications.
- **Full scale experimental validation.** We give a proof of concept of a hybrid heterogeneous implementation of RTM for cluster containing manycore architectures and give measurements on Stampede. We give strong scalability tests up to almost 16400 threads on 5000 heterogeneous core and compare with other implementations of RTM.

In the following paragraphs, we give the outline of the thesis.

We start with an introductory chapter 1 where we describe the challenges brought along the Exascale Era. It concerns hardware, software and algorithms levels. Due to constraints on power consumption and cost in order to ensure the feasibility of an Exascale machine, we can expect revolutionary changes on the hardware. As a consequence and to guarantee the efficient use of the available resources, software and algorithms will also know great modifications. The way around study is also important. The existing algorithms and software stack may have constraints that need to be considered while building a new system. This mutual interaction between all levels constituting a high performance machine is called the co-design process and is considered as a key step towards the Exascale.

In the following chapter 2, we give an overview of the different numerical methods used in seismic imaging. These various methods respond to different needs in terms of the image resolution, time to solution, the available computational resources, the accuracy of the input models, etc. We provide in chapter 2 the equations governing wave propagation in isotropic and anisotropic media and the resulting approximations using the finite-difference time-domain (FDTD) approach.

The third chapter 3 is a deep insight on performance study and characterization. We give state-of-the-art performance modeling techniques and give a first model for FDTD applications considering theoretical peak performances. We then go through state-of-the-art optimizations approaches applied to FDTD and give

an analytical model for cache blocking based on previous works. We also use a sampling-based tool Adaptive Sampling Kit (ASK) in order to tune input and optimization parameters for FDTD. We consider the grid size, number of threads, cache blocks sizes, the order of the numerical scheme and the variant of the kernel where we apply some loop modifications or not.

Chapter 4 applies the Roofline model introduced in chapter 3 to FDTD applications. Optimizations are also applied gradually on our applications and demonstrate their impact using the Roofline model. We implement a model of the DRAM traffic based on the data reuse histogram. We have an updated formulation for cache blocked versions since we had an over-estimation using the initial model in this particular case.

In chapter 5, we implement heterogeneous and hybrid version of the FDTD applications. We study the impact of the increasing parallelism and heterogeneity on these application through the use of Intel's Xeon Phi as representative of a future Exascale node.

The last chapter 6 is a feasibility study applied to an implementation of the Reverse Time Migration (RTM) application. We extend the performance model to communications and I/O for large scale systems. We also make prior modeling for a heterogeneous system containing Xeon Phi which enabled us to identify the expected bottlenecks on such systems. We then give measurements on the machine Stampede.

Finally, we conclude by summarizing the different contributions and lessons learned and we suggest some future work.

CHAPTER 1

Motivations

1.1 Exascale Challenges

After the race to reach the Petascale performance and the tremendous impact it had on the current HPC systems, scientists are facing another challenge with the Exascale machines expected for the 2018-2020 time-frame. An Exascale machine delivers 10^{18} floating point operations per second, that is to say 1000x the performance of the most powerful machines currently available. We think that we have come to a turning point of computing technology evolution since merely augmenting the number of the computing units in a machine is unlikely to ensure an exaflopic performance. High performance computing (HPC) systems including the computing, storage, interconnect and cooling systems need to be reviewed for the Exascale era based on constraints fixed by the HPC community in order to ensure the feasibility of these future machines from an economical and environmental point of view. As a consequence, power and cost factors will be determining characteristics in building systems for the Exascale.

Challenges will affect all conception levels of a computing machine from hardware to software stack and applications. Hardware level includes the node architecture, interconnect and memory systems. The software stack concerns the operating systems, compilers, runtimes and programming models. Applications and algorithms need to evolve in order to leverage the new hardware features offered by the system. Resiliency and errors management are new constraints imposed by the increasing number of cores and limitations on power consumption. The current approaches of fault tolerance will be inefficient. More practical recovery mechanisms need to be developed.

Another aspect of the Exascale transition is the co-design process. HPC experts seem to agree that efficiently building such systems is conditioned by the collaborative effort conducted by the previously cited communities [Kogge et al., 2008; Ashby et al., 2010; Dongarra et al., 2011]. Interacting mutually will help architects to define new architectures that can be used efficiently by the applications and algorithms communities. On the other hand, applications and software stack need to be adapted to the constraints of the future hardware.

In the following paragraphs, we are going to detail all these challenges for the

hardware, software and application communities and will describe the co-design process.

1.1.1 Hardware Challenges

We present the hardware constraints we need to handle when designing an Exascale machine. The role of these constraints is to guarantee a feasible and maintainable system. We start with the energy efficiency challenge since it is expected to drive the design of nodes, memory systems and interconnects. Resiliency is another aspect that we should consider in an Exascale system. The recovery time should be maintained the shortest possible in order to limit the system's unavailability.

Table 1.1 gives the characteristics of the top two machines, Tianhe-2 and Titan, according to the ranking in the Top500 [1] list of June 2014. In this 43rd edition of the Top500 project, Tianhe-2, installed by China's National University of Defense Technology, is the most powerful system. It uses Intel Ivy Bridge sockets and Intel Xeon Phi coprocessors. On the other hand, Titan, ranked second, is a Cray system installed in the Oak Ridge National Laboratory based on AMD Opteron Interlagos sockets and NVIDIA Tesla K20x GPUs. The features of these two machines and their performances summarized in table 1.1 will be used as a reference for the comparison with the potential Exascale machine as described in the upcoming paragraphs.

	Titan	**Tianhe-2**	
System peak	27	54.9	[PFlops]
HPL benchmark	17.59	33.86	[PFlops]
System nodes	18,688	16,000	
System cores	560,640	3,120,000	
System memory	0.710	1.4	[PByte]
Node performance	1.4	3.4	[TFlops]
Node concurrency	30	195	
Node memory	32/CPU + 6/K20	64/CPU + 8/Phi	[GByte]
Power	8.2	17.8	[MWatt]

Table 1.1 Characteristics of the top 2 machines according to the ranking of top500 in June 2014.

The values of power consumption in table 1.1 corresponds to the power needed by the processors, memory systems and interconnect only. Cooling is not included in these values. We also give the theoretical system peak performance and the effective peak performance measured by the High-Performance Linpack [2] (HPL) benchmark [Dongarra et al., 2003].

[1] www.top500.org
[2] www.netlib.org/benchmark/hpl

1.1.1.1 Power Management

For this paragraph, we consider the peak performance of Tianhe-2 and Titan machines measured by the HPL benchmark and their power consumption as reported in table 1.1. Exascale machines are expected to deliver a HPL peak performance equal to 10^{18} floating-point operations per second. A linear extrapolation of the power consumption of Tianhe-2 and Titan estimates over 500 MW of power only for processors, memory and interconnect to attain exaflopic performance. This prohibitive value proves that extending the actual petaflopic systems while only adding computational resources is unlikely to deliver a maintainable system. For obvious economical and environmental reasons, minimizing the power consumption of Exascale systems is a major concern.

The US Defense Advanced Research Projects Agency (DARPA) launched in 2010 the Ubiquitous High Performance Computing (UHPC) program [DARPA, 2010; Carter et al., 2013]. It aims at building computing systems that respect an energy efficiency constraint fixed to 50 GFlop/Watt for 2018 time frame. Therefore, power consumption for Exascale systems will be equal to 20 MW without counting the cooling consumption. As recommended by [Shalf et al., 2011b; Kogge et al., 2008], exceeding this value is likely to hinder the system performance and it can have a great influence on the machine cost.

Given the limitation on power consumption of an Exascale machine, a floating-point operation requires 20 pJ of heat on this machine [Borkar, 2013]. Table 1.2 compares the amount of heat required to perform floating-point operations and DRAM reads on current machines with expected values in Exascale. We await for a reduction by at least a factor of 4 to ensure energy efficiency.

Figure 1.1 presents the evolution of power consumption considering only floating-point operations for top machines since June 2008.

	2011	**2018 Expectations**
DP FMA Flop	100pJ	20pJ
DP DRAM Read	2000 pJ	1000pJ
Local interconnect	7500 pJ	1000pJ
Cross System	9000 pJ	1500pJ

Table 1.2 From Shalf et al., 2011b. Data movement cost in term of power consumption.

Contribution of data movements between memory hierarchies to power consumption in current systems is high as shown in the listing 1.2. It is even higher when it comes to use the network in order to move data from one node to another. Predicted costs for Exascale machines are expected to increase. Reducing data movements is consequently mandatory to reduce the power consumption. Leveraging data locality mechanisms is necessary as it enables the reuse of the data fetched in the faster memory levels. It is also important to rethink applications in order to reduce communications between nodes and therefore limit the use of the network. This results in a decrease of the demand on bandwidth and the impact of latency of both memory and network.

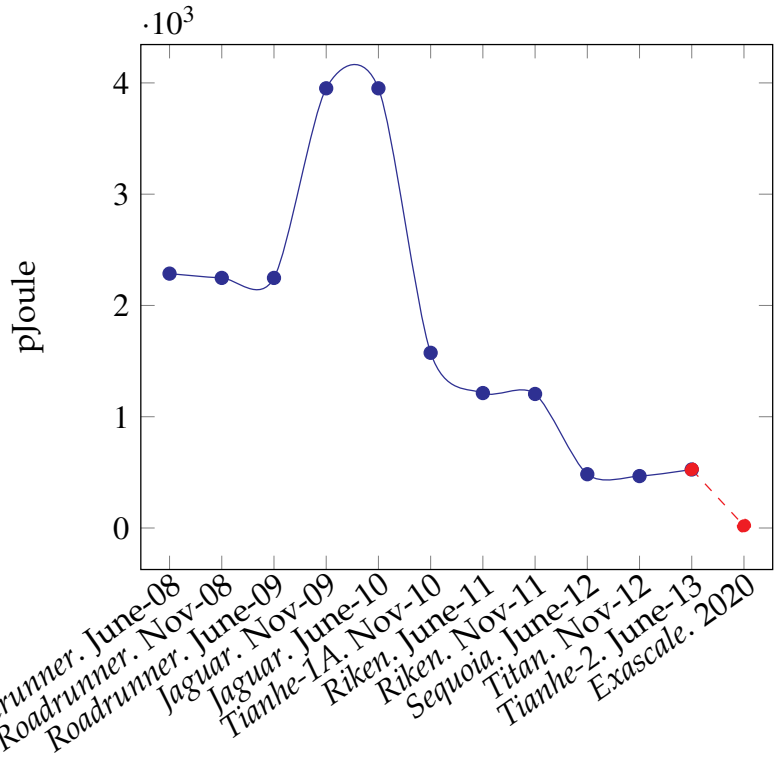


Figure 1.1 Cost of a floating point operation in term of energy consumption for machines ranked first in top500 from June 2008 to June 2013.

This leads us to the following paragraph where we depict the challenges related to memory systems design while maintaining low power consumption.

1.1.1.2 Memory Systems

As described in figure 1.2, memory hierarchies in computing systems are mainly characterized by their latency, bandwidth and capacity. The cost per bit of memory is also an important feature in computing systems since it can guide the trade-off between capacity and speed. The figure 1.3 illustrates the evolution of dollars/MByte and dollars/MFlop rates from 2002 to 2011 and shows a clear drop of flops cost compared to the modest decrease of bytes cost. This explains the gap between processing units and memory which results in the inefficacy of a wide range of applications bounded by memory bandwidth or latency. This gap is expected to be even wider in Exascale systems.

A machine balance is a metric expressing the number of bytes transferred per a floating-point operation. The value of this metric gives an idea on the behavior of an application on a given architecture. Depending on the arithmetic intensity of this application, we can figure out if the memory bandwidth is going to be a bottleneck or if it is the processing units. Rethinking the current memory systems is important to maintain a low machine balance in Exascale and ensure a larger number of applications running efficiently on these machines.

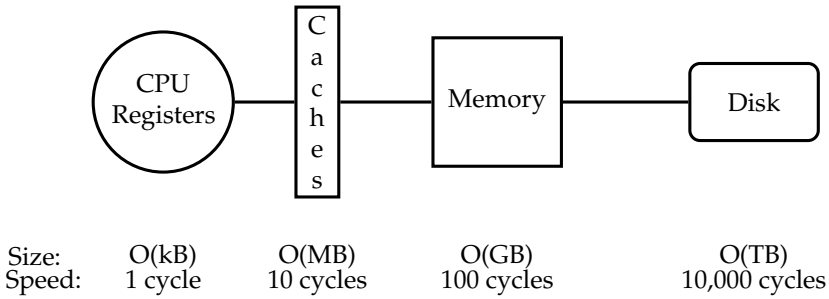


Figure 1.2 Memory system hierarchy.

In the following paragraphs, we give possible solutions in order to attenuate the limitations on memory bandwidth, latency and capacity.

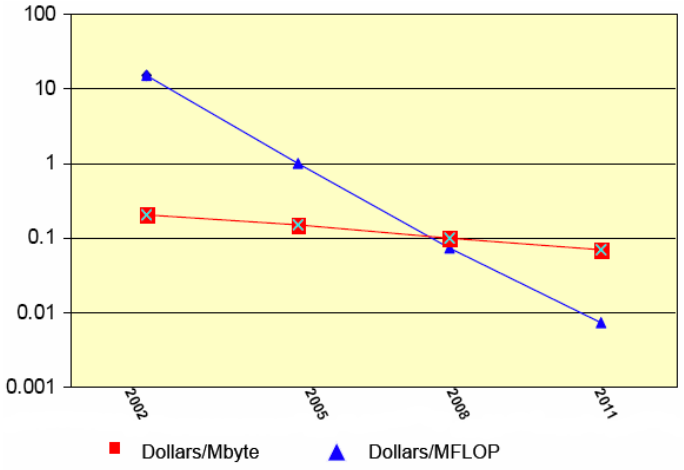


Figure 1.3 From Dave Turek, 2009. Reduction of Flops and Bytes costs.

Packaging Packaging of the memory and the processing unit can greatly influence the bandwidth rates. Re-defining the architecture of current systems may result in an improvement of the bandwidth and thus increase the overall system performance. The figure 1.4 is a roadmap proposed by Camp et al., 2010 describing trends in CPU and memory packaging likely to surpass the hardware limitations we are facing today in increasing the bandwidth. Ultimately, a 3D stacked packaging is expected to deliver a bandwidth greater than 1 TB/s at the cost of challenging integration issues. [Coteus et al., 2011; Loh, 2008; Woo et al., 2010]

Memory Bandwidth As the number of cores per node is expected to increase as we are moving towards the Exascale, the memory bandwidth will clearly be a bottleneck that can greatly impact the application performance. New packaging techniques as described previously give solutions to solve the bandwidth issue on

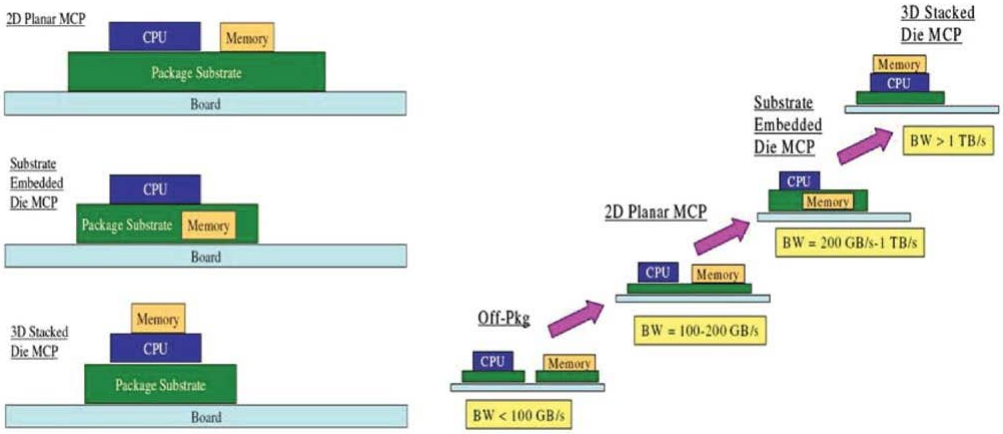


Figure 1.4 From Camp et al., 2010. Roadmap for memory and CPU packaging in order to respond to the bandwidth demand.

the hardware side. Data locality is another solution to get around the bandwidth limitations on the applicative side which implies some modifications of the existing implementations.[Stevens et al., 2009].

Memory Capacity Figure 1.3 illustrates the evolution of the cost of floating point operations and bytes in term of dollars. In 2011, flops are 1000 times cheaper than in 2002. This is not the case of bytes barely 10 time cheaper in 2011 than in 2002. Increasing of the aggregate memory capacity contributes greatly to the cost of the whole system (see figure 1.3). Taking into consideration the economical constraints for Exascale systems, this tendency observed on figure 1.3 is expected to continue on Exascale systems, resulting in a great augmentation of the number of FPU compared to the memory capacity. Shalf et al., 2011c estimate an improvement factor of 100x in memory capacity compared to a factor of 1000x in the system peak floating point performance.

1.1.1.3 Node Architecture

Figure 1.5 describes the scaling of general-purpose processors from 1975 to 2012. We notice that since 2005 the frequency and the single-thread performance are witnessing a slow-down of their evolution. On the contrary, the number of cores is increasing greatly starting from 2005. It is the response to the decrease of the single thread performance and frequency. An extrapolation to the Exascale systems and taking into consideration to power limitations described in paragraph 1.1.1.1, the number of cores will continue to increase and as a consequence we expect unprecedented changes of the node architecture. In paragraph 1.1.1.2, we considered the widening gap between memory system and the processing units. A possible solution in order to hide latencies due to this gap is the use of multi-threading as it permits to reschedule a second thread while the first one is, as an example, waiting for data [Borkar et al., 2011].

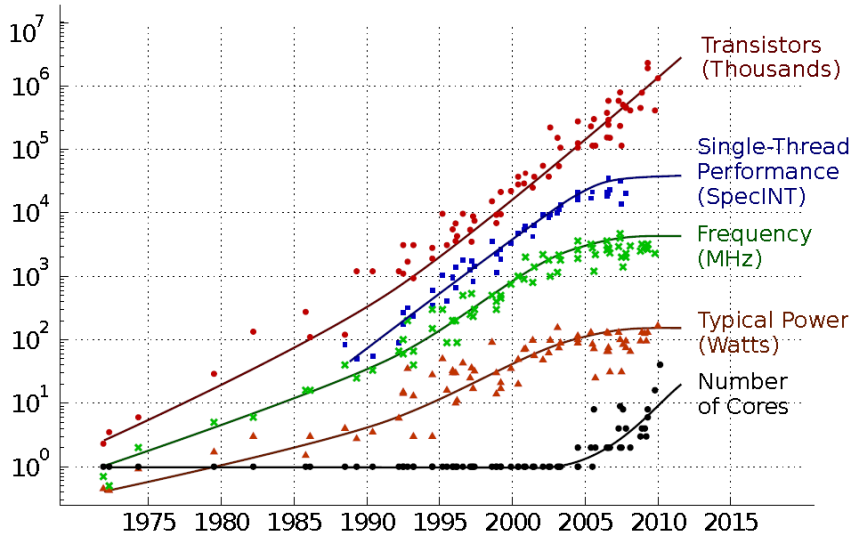


Figure 1.5 From Batten, 2010. General-purpose processors trends in terms of power consumption, number of transistors, frequency and number of cores.

Parallelism It is reflected by the increase of the number of computing units which are expected to double every 18-24 months [Hall et al., 2011]. As a consequence, we need to compensate the need to higher compute capacities by the mean of parallelism at the node level using many cores and threads.

This leads us to two major trends in the node architectures and consequently the whole system as described in Kogge et al., 2008. We can expect to have heavyweight nodes with relatively restricted number of cores but having advanced features. On the other hand we can have lightweight nodes with a high number of cores with less sophisticated capabilities but offering the possibility to use parallelism to compensate the latencies.

Heterogeneity It is another opportunity to enhance computational capabilities of the node while maintaining a reasonable power consumption. Current accelerators are connected to the host via a PCIx interconnect and have separate memory spaces. Therefore data movements between the two devices are not supported explicitly and data placement should be done at a high level in the application in order to minimize the cost of data movements.

AMD fusion project is an example of the willing to get rid of the PCIx interconnect and to homogenize the memory accesses in order to facilitate the programming of these architectures while maintaining a lower bound on power consumption.

1.1.1.4 System Resiliency

The increasing number of cores and the need to use a threshold for power may be the reason for frequent fault accidents. The recovery time, known as the Mean

Time To Repair (MTTR), is expected to increase in an Exascale machine. Fault accidents and time needed to recover them are also expected to grow. Resiliency can be permanent and in this case it occurs in the hardware or transient and it happens in the software level. In this last case, It can be recovered relatively quickly.

Mechanisms such as I/O checkpointing to prevent these failures are extremely important but they should not be intrusive and degrade the application performance.

1.1.1.5 Network Interconnect

Many applications need domain decomposition techniques in order to remedy to shortage in memory capacity on the computing nodes.

This implies massive utilization of the communication network. Consequently, latency and bandwidth of the network have great impact on performance on applications and neglecting the I/O may induce issues for data intensive applications.

For Exascale machines, the issue of I/O bounded applications will persist and even worsen. Overlapping I/O with computations and the use of asynchronous accesses will be probably used but they won't be enough in reducing the overhead generated. New technologies such as Photonics introduced by Intel and Fujitsu are needed to alleviate this issue [Coteus et al., 2011; Camp et al., 2010].

1.1.1.6 Potential Exascale System

As a summary, the potential Exascale systems can have two possible configurations as described in the DARPA's report on Exascale transition [Kogge et al., 2008].

- **First Lane** where cores are expected to be complex. These systems will be the earlier version of Exascale machines and are expected around 2018. This configuration will induce reduced number of nodes.
- **Second Lane** where cores are rather lightweight and their number will increase per compute node. This is the long term version of an Exascale system where thread level parallelism will be an important mechanism to hide latencies.

Table 1.3 summarizes of the potential characteristics of an Exascale system as described in the technical reports of the US Department of Energy (DOE).These are a preliminary results subject to modifications with the upcoming machines [Kogge et al., 2011].

A profusion of architectures is emerging in order to study the possibilities of saving energy while performing the required floating-point performance to reach an exaflopic performance. The Echelon architecture implemented by Nvidia and described in Keckler et al., 2011 is an example of architectures that aim to limit energy consumption to 20 pJ per floating-point operation. We can also name Runnemede an Intel architecture that is a response to the UHPC project. This architecture leverages a co-design process in order to respect the architectural constraints

	Tianhe-2	1st Lane	2nd Lane
System peak	54.9 Pf/s	-	-
HPL benchmark	33.86 Pf/s	1 Ef/s	1 Ef/s
System nodes	16,000	O(10^6)	O(10^5)
System cores	3,120,000		
System memory	1,4 PB	32-64 PB	32-64 PB
Node performance	3.4 TF/s	1 TF/s	10 TF/s
Node concurrency	195	O(10^3)	O(10^4)
Node memory	64 GB CPU + 8 GB Phi	32-64 GB	0.32-0.64 TB
Power	17.8 MW	20 MW	20 MW

Table 1.3 Potential Exascale systems.

fixed by DARPA for a feasible Exascale system. Another example of architecture enabling energy efficiency in order to facilitate the transition to Exascale systems, we can also name the Green Wave architecture introduced in Krueger et al., 2011 to study hardware/software co-design for seismic modeling applications.

1.1.2 Software Challenges

Exascale is not only a matter of hardware features. It also implies tremendous modifications of the current software stack as discussed in [Barrett et al., 2012; Dongarra et al., 2011]. We are going to expose the possible modifications affecting the operating systems, compilers, runtimes and programming models and debugging tools in order to prepare them to the expected hardware constraints in Exascale systems. We need to take into account the increase of concurrency and parallelism, the expected limitations on main-memory and networks in terms of bandwidth and latency, the power efficiency requirements and the resiliency considerations. This is an important task to undertake to ensure an efficient use of the hardware.

1.1.2.1 Operating Systems

Operating systems are important components of the software stack on the current machines. On Exascale systems also, their role will be essential in maintaining an efficient use of the hardware resources since inadequate operating systems can result in dramatic impact on performance.

Two approaches are possible either rewrite from scratch a new operating system or upgrade an existing implementation based on the expected architectural features of the future systems. In both cases, we should take into considerations few constraints such as the increasing concurrency and the need to hide the system latencies using many threads per core. The execution of a single task using many threads will be therefore a common technique. We need also to think of new mechanisms to unburden the shared resources between the computing cores. The memory bus, for example, can cause great performance issues.

1.1.2.2 Programming Models & Runtimes

As the number of cores per node are expected to increase greatly, we need to maximize parallelism in applications and have a reliable runtime in order to ensure an efficient scheduling of tasks among the workers. This important feature needs to be handled transparently by the programming model. Developer's concern is focused on programming the scientific application.

For Exascale, we might also need more than a programming model to handle the increasing number of cores per node and ensure interoperability between them. As an illustration, we can consider hierarchical programming where we perform communications using message passing interface MPI and computation using a shared memory model such as openMP. The combination of these two models depend on the needs of the application. We can have a master-slave model where the MPI communication is performed by a single thread, the master, and the openMP threads compute. An other configuration consist of multiple communications performed by multiple threads simultaneously. For this case, the MPI implementation should be thread safe. On the other hand, MPI implementations should support a more sophisticated manner to identify threads rather than tags which cannot offer sufficient flexibility for all applications.

Partitioned Global Address Space (PGAS) programming models such as Co-Array Fortran (CAF) represent an interesting opportunity. Open64 compiler supports an implementation of CAF. [Eachempati et al., 2012] is a proof of concept that this programming model can easily included in an industrial code. Scalability and performance reported on Reverse Time Migration give an encouraging start to adopt these programming models in industry but CAF is more suitable for applications with regular communication patterns. I/O are also a challenge to consider as they represent one of major bottlenecks for these data-intensive applications

1.1.2.3 Compilers

Compilers have a great impact on applications performance since they guarantee the efficient use of the underlying architecture. Low level optimizations such as loop modifications like padding and tiling, vectorization, software prefetching are architecture-dependent optimizations and are fastidious implementing within the application code. For Exascale, we will be facing heterogeneity on one hand in order to profit from the low power consumption of some devices such as accelerators. On the other hand, we need to ensure the efficient use for every single architecture since they can have different vector widths and different instructions.

1.1.3 Algorithmic Challenges

As section 1.1.1.1 suggests, data movements in Exascale systems are expected to be extremely expensive in terms of pJoule/Byte. The main reason is the growing gap between arithmetic units and the memory system and the increasing power consumption that it will generate. As a consequence, neglecting data locality may hinder extremely the performance of applications on Exascale systems. Algorithms need to enhance data locality in time and space in order to reuse as much

as possible data fetched in caches and avoid unnecessary memory traffic.

Extrapolations to Exascale predict that floating operations are going to be available in abundance and compared to memory traffic they will be almost free. We can use these arithmetic units to introduce more complexity in our computations when overlapped with communications, we can hide network latencies. [Dongarra et al., 2012]

Communication-avoiding Algorithms. Many applications are bounded by the time spent in communications and in memory traffic. Since the gap between the processing units and memory systems is exponentially increasing and since the latency on network can be penalizing, we need to limit the data movements for the current Petascale systems and more urgently for the Exascale machines. Communication-avoiding algorithms aim to minimize the impact of these communications on performance by minimizing the number of messages transferred and their volume. They also can rely on redundant computations. These algorithms are mainly dense and sparse linear algebra algorithms which require the implementation of new numerical methods while ensuring the stability of these new solutions [Khabou, 2013]. For example, the Strassen algorithms are a communication-avoiding variant of matrices multiplications [Lipshitz et al., 2012]. Apart from the impact on performance, minimizing the data movements in these new algorithms will reduce the power consumption compared to the usual implementations.

Fork-join Model. This technique is extensively used in scientific applications. Programming models such as OpenMP permit to use multi-threading capabilities in order to accelerate the computing loops. As the number of cores per node in Exascale machines is expected to greatly increase, the overhead due the initialization of these parallel sections and due to the synchronization will consequently increase and deteriorate the overall performance of applications.

Load Balancing. As heterogeneous architectures are expected to be extensively used in Exaflopic machines, we need to implement algorithms that enable load balancing in order to avoid stalls that are inherent to this heterogeneity. In some cases, handling the load balancing may require the implementation of hardware specific modules and even the modification of the whole algorithm in order to introduce new mechanisms such as scheduling and tasks queuing. The aim is a better management of available resources.

Multi-scale algorithms also require load balancing. We can name multi-grid algorithms applied to seismic modeling. The mesh refinement is performed only in the area where the simulation needs more precision.

Auto-tuning. This refers to a set of tools used to determine the best combination of optimization factors for an application class depending on the targeted architecture. It involves the number of threads, cache blocking factors, padding length, etc. Since heterogeneity is inevitable in Exascale systems, machines will be more

complex. As a result, the optimization process will be tedious and time consuming task which explains the need to auto-tuning tools in order to use efficiently the architecture.

Performance Metrics. New metrics are emerging as we are preparing Exascale systems. Considering the constraints on power consumption, usual performance metrics reporting the number of operations performed per a unit of time for example will not be considered as the unique reference on the efficiency of a given application. Metrics such as pJoule/Byte or pJoule/operation are going to be determinant in illustrating an application behavior on Exascale systems [Camp et al., 2010].

1.1.4 Co-Design

The Department of Energy (DOE) Exascale Computing program is an initiative to gather efforts in hardware and software areas and work jointly in order to build efficient Exascale systems. This is the opportunity to establish a co-design process applied to Exascale. In this study, the applications design take part in the improvement of the hardware. Interaction between developers and architects is a key element to elaborate new architectures that take into consideration algorithmic requirements. On the other hand, an Exascale system is expected to introduce various changes on the hardware. It may concern the node architecture, interconnect and the cooling of the whole system. Consequently, algorithms will need to adapt in order to efficiently run on these architectures. This includes data management, the use of memory system and the arithmetic-unit and IO bandwidth. The figure 1.6 is a high level illustration of co-design. For applications side, we need to consider the algorithms, the underlying performance models and the implementation. On the system side, we need simulators, building of new micro-architectures and interconnects. We also need the development of an adapted software stack that enables an efficient use of the hardware.

The Co-Design for Exascale (CoDEX) project [Shalf et al., 2011a] is a response to the DOE initiative. It combines a bunch of tools that range from cycle accurate simulator to node architecture to an extrapolation of memory and interconnect for the Exascale. These tools allow a software/hardware co-design illustrated in [Krueger et al., 2011] for an energy study applied to seismic modeling.

In order to facilitate the co-design process, applications are not studied as a whole. We instead consider lighter versions of these applications ranging from simple kernels capturing the computational core of the application to more sophisticated versions gradually including physic considerations and I/O operations since they are extensively use in real applications. The classification of the different versions leading to real applications are explained in table 1.4 as described in [Shalf et al., 2011c] where the word *surrogate* was used to designate this light versions. They are extremely useful in a co-design process since they permit to highlight the architectural features impacting the performance at each level.

For the feasibility study in Exascale applied to seismic imaging applications, we will rely on these surrogates in order to overcome the burden in production

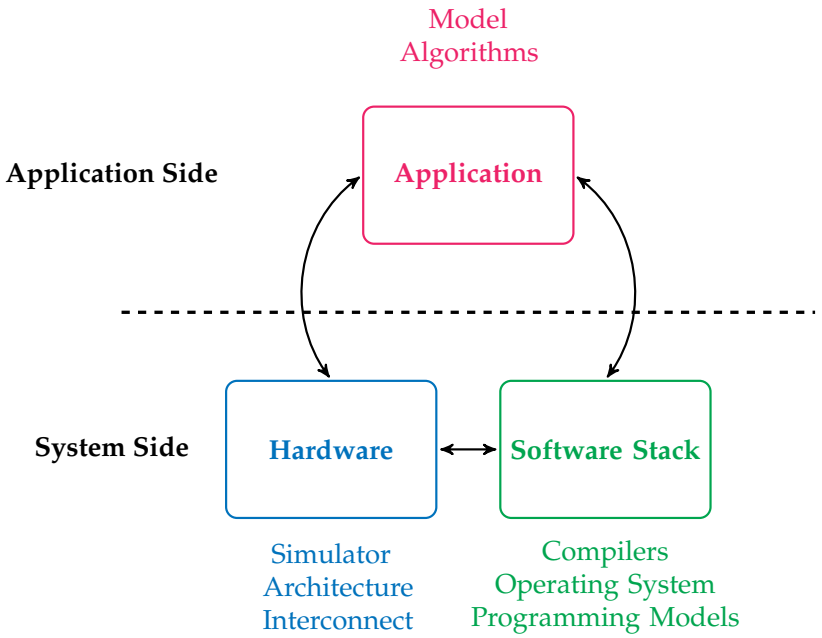


Figure 1.6 High level description of the co-design process. We emphasize on the interaction between the application side and the system side in order to enable the design of the adequate hardware and algorithms.

Surrogate	Description
Compact application	Small app with fewer features and simplified boundary conditions relative to a full app
Mini-application	Small, self-contained program that embodies essential performance characteristics of key apps
Skeleton application	Program that captures an app's control flow and communication pattern; can be run only in a simulator
Proxy application	General term for all other surrogates
Mini-driver	Small programs that act as drivers of performance-impacting library packages
Kernel	Program that captures an algorithm's node-level aspects

Table 1.4 From Shalf et al., 2011c. Definition of application surrogates.

codes. We will start with kernels highlighting the computational behavior of imaging applications on the node level. We will particularly focus on an application called Reverse Time Migration (RTM) as it is widely used by Oil & Gas companies. We will then consider a compact-application version of RTM to facilitate the port-

ing on a representative architecture of the trend in Exascale systems. It's the Intel Many-Integrated Core (MIC) architecture. This will enable us an extrapolation to a larger scale system with representative features of an Exascale machine.

1.1.5 Previous Feasibility Studies

In [Bhatele et al., 2011] and [Gahvari et al., 2010], we can find feasibility studies of Exascale systems based on extrapolations of hardware features and performance models of representative application classes commonly used in scientific computing.

Gahvari et al., 2010 conducted a study on FFTs and basic geometric multigrid. Scalability was studied in both cases for the Exascale and an evaluation of the requirements in terms of latency and memory bandwidth on such systems was presented based on performance models of these applications.

A similar work was presented by Bhatele et al., 2011 where the applications considered are molecular dynamics, cosmological simulations and finite element solvers. The study concerns the weak and strong scalability of these applications on Exascale machines. It also considers memory requirements and the volume of communications compared to computation requirements.

Further co-design studies can be found in [Czechowski et al., 2011a], for example, where authors revisited simple metrics commonly used by the HPC community such the balance principle, Little's law, Amdahl's law, etc.

The co-design principle can also be applied to the energy efficiency challenge. Krueger et al., 2011 give a study that aims to minimize the power consumption. They consider a seismic imaging application called Reverse Time Migration RTM. Co-design studies also focused on other applications such as FFT [Czechowski et al., 2011b; Czechowski et al., 2012], molecular dynamics, finite-element solvers and cosmological simulations [Bhatele et al., 2011].

1.2 Geophysical Applications

The establishment of the theories governing geophysical phenomenons started since 1960. As the methods developed are mainly compute intensive, the computing technologies evolution in the 80's was extremely promising since they made possible their effective application in a real context such as seismic exploration. They also prove to be beneficial to study earthquakes and to image the earth layers in order to facilitate the localization of hydrocarbon deposits.

As a consequence, the evolution of the seismic applications is historically related to the evolution of computers and afterward HPC systems. The growth of the compute capacities and the amount of the data they handle permitted the utilization of new numerical methods with complex physics and the exploration of wider area with higher resolutions.

Given the actual HPC context, seismic applications constitute an important class that we need to consider in the co-design process. Classification and characterization of the major approaches in seismology using modeling is an interesting

task that permits the extrapolation of the performance on different architectures and herein determines which methods are more suitable for Exascale machines.

Figure 1.7 depicts the resource requirements as a function of the complexity in terms on computation. Resource needs increase exponentially as the media studied is complex and as the method is compute demanding. The figure is also an illustration of the trends in seismic imaging. It shows that the increase in compute capacity permits to tackle geophysical complexity easily. It enables more accurate modeling of wave physics, higher grid resolution and wider frequency content.

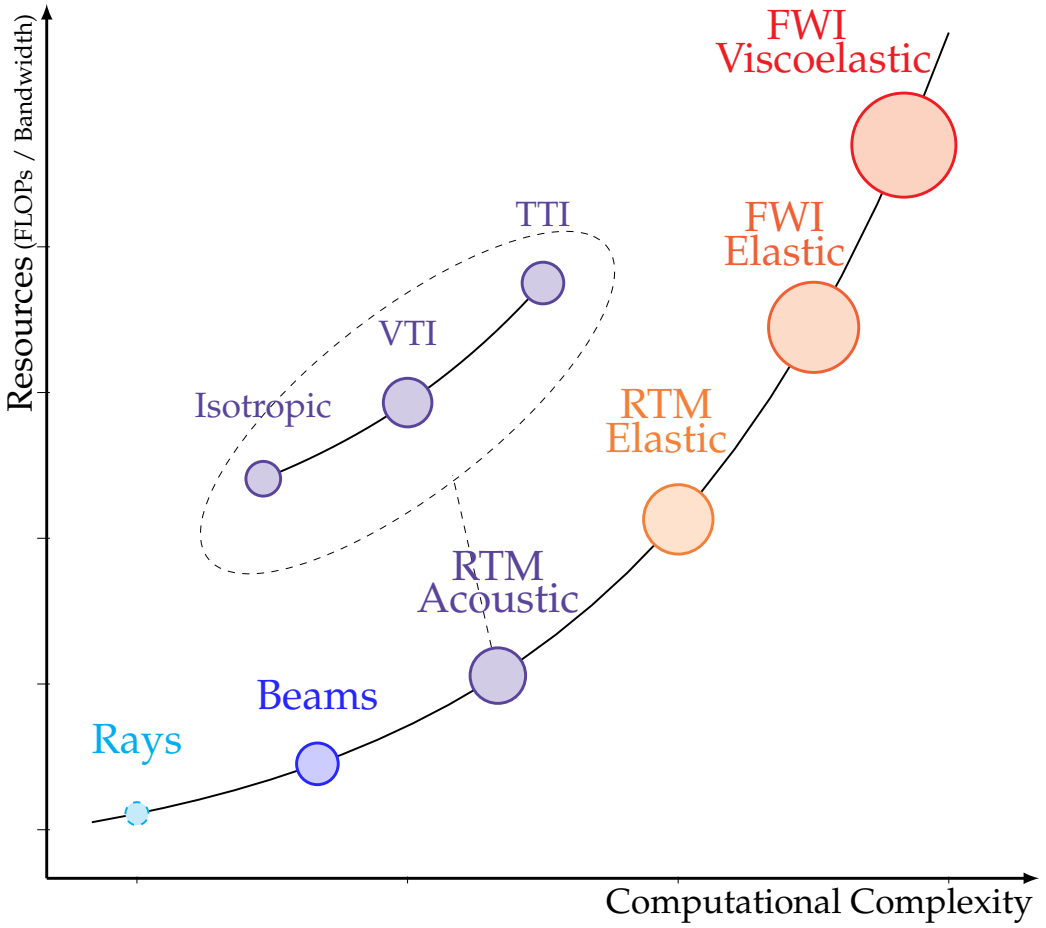


Figure 1.7 Seismic imaging applications.

1.2.1 Seismic Imaging Applications

Exploration seismology techniques are widely studied by the scientific community. One can find detailed descriptions of these techniques in a multitude of studies such as [Sheriff et al., 1995; Biondi, 2006] to name few. In the following paragraphs, we define some of these seismic imaging applications.

1.2.1.1 Seismic Modeling

Seismic modeling applications build a synthetic image of the earth based on a model of the velocity making assumptions on the physical properties of the medium studied. In order to approximate the wave equation, seismic modeling can rely on a multitude of numerical methods such as integral methods, asymptotic methods and direct methods presented in chapter 2.

1.2.1.2 Reverse Time Migration

The Reverse Time Migration (RTM) is an application widely used by Oil and Gas companies in order to build an image of the earth crust. Figure 1.8 depicts from a high level point of view the 3 steps in RTM.

We start with a forward modeling using an initial model of the velocity. Then, we perform a retro-propagation of the wave equation using the recorded data. It is the backward modeling. Finally, we produce the final image using a cross-correlation applied on the computed values in the previous steps.

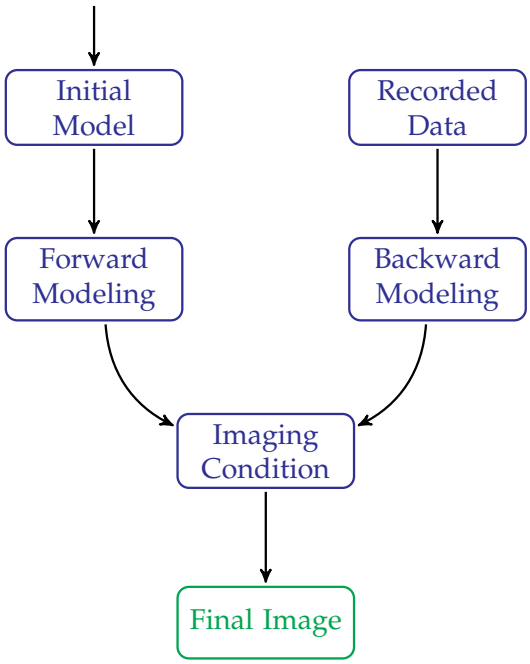


Figure 1.8 Reverse time migration work flow.

1.2.1.3 Full Wave Inversion

The Full Wave Inversion (FWI) aims to build a velocity model of the subsurface studied. As described in figure 1.9, we compute the misfit between the computed wavefield and the observed data and iterate on the corrected data until we converge.

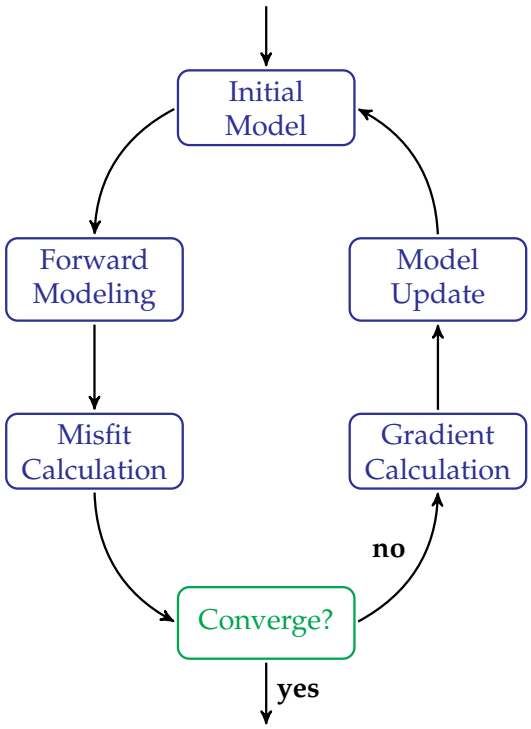


Figure 1.9 Full wave inversion work flow.

1.2.2 Seismic Exploration Work Flow

Seismic exploration surveys are long processes that aim to have a precise localization of hydro carbon deposits in a given area which enables quick and precise drilling. This is an important process for Oil and Gas companies because it permits to significantly save time and money.

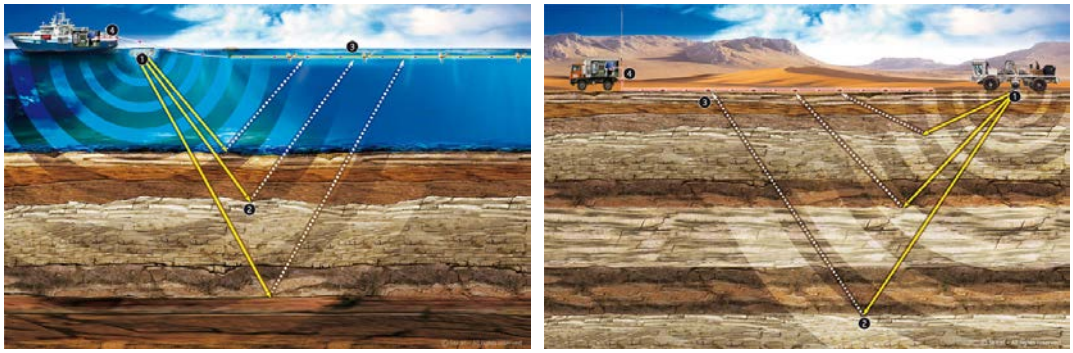
The data acquisition during these surveys is followed by other steps before obtaining a final image. It consists in : velocity modeling, wave field migration and interpretation.

1.2.2.1 Data Acquisition

Data acquisition is performed during exploration campaigns. These campaigns occur whether on land or in marine. The approach is the same in the two environments but involve different tools. The process consists of four steps.

The creation of a wave propagation using seismic vibrators or explosive sources in land or compressed air guns in marine. The wave generated propagates into the earth crust. At discontinuities, reflections occur. The reflected waves are recorded using geophones in land and hydrophones in marine. Preprocessing is the last step. It aims at preparing the collected data during the survey. It consists in removing the noise contained in the raw data since it can interfere in the final image and result in misinterpretation. Figure 1.10 explains the different steps required

during seismic acquisition in marine and in land. The previous four steps are illustrated in both cases.



(a) Marine seismic acquisition **(b)** Land seismic acquisition

Figure 1.10 Seismic acquisition steps in marine and in land: 1) acoustic source emits energy 2) seismic waves propagate in subsurface layers 3) reflected waves are recorded by geophones or hydrophones 4) raw data is processed. Courtesy of Sercel, 2014.

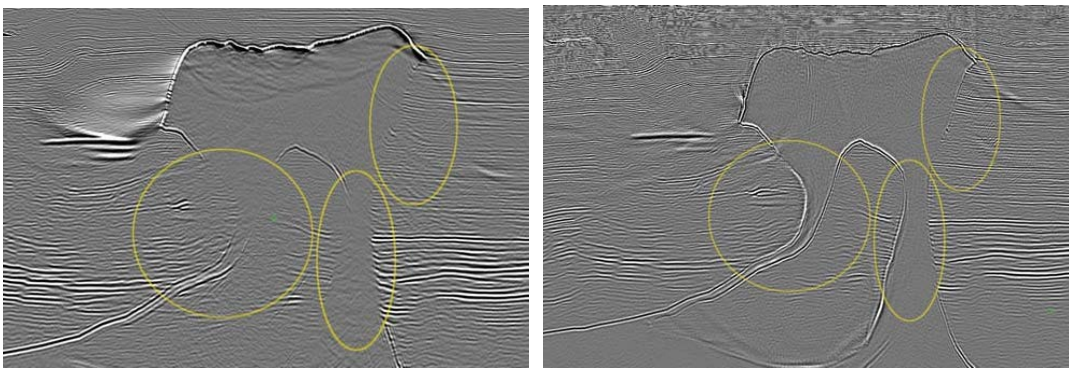
1.2.2.2 Migration/Imaging

Migration places geometrically the discontinuities of the subsurface encountered during the wave propagation. It aims at reconstructing an image of the crust. The migration complexity depends on the nature of the area explored. Therefore, as described in [Sheriff et al., 1995], we have the following approaches:

- *Time migration.* It produces an image function of time. It is used when contrasts are small and when the geological structure is simple. This explains why it is mainly used for sedimentary basins.

- *Depth migration.* It gives an image function of depth. It's more appropriate for complex structures with sharp slopes. It is also used in subsurfaces containing salt domes. As described in [Farmer et al., 2009], depth migration can be ray-based like Kirchhoff migration and beam migration. These approaches are used for steep structures but are inefficient when strong velocity variations are encountered. Wave equation migration techniques are another family of depth migration. We have the one-way wave equation migration which is unable to image accurately salt structures with strong dipping flanks. This limitation is removed in the two-way wave equation migration which is more efficient in imaging steep structures and extreme velocity variations especially on the surfaces neighboring salt structures. Figure 1.11 illustrates the one-way and two-way migrations using the BP 2004 benchmark [Billette et al., 2005]. The yellow circles depicts the weaknesses of the one-way migration next to the salt dome flanks.

We call seismic trace the data recorded by the receivers during the recording time of the seismic acquisition. The migration can occur before the summation of



(a) One-way wave equation migration: split-step Fourier plus interpolation.

(b) Two-way wave equation migration: reverse time migration.

Figure 1.11 Images showing the difference of imaging for two different depth migration algorithms. Courtesy of Farmer et al., 2006.

these traces and it is called pre-stack migration in this case. Or, it can be applied on the result of their summation and then we call it post-stack migration. Pre-stack migration gives more accurate results but requires more computations compared to the post-stack migration.

1.2.2.3 Interpretation

The image produced by the migration algorithms need to be studied in order to localize the possible hydrocarbon deposits. The quality of the image depends on the algorithm used for the migration.

1.2.3 Challenges in Seismic Imaging Applications

Challenges in seismic imaging applications are related to the nature of the subsurface studied and the available resources to give a precise image of the subsurface in a reasonably short time.

Time-to-solution vs cost-to-solution Time-to-solution is the time needed for an application to converge while cost-to-solution includes the necessary resources needed in terms of compute capabilities, energy, maintenance, etc. Among major concerns when it comes to production codes, such as RTM and FWI, is to find a trade-off that minimizes both time- and cost-to-solution.

Data management and I/O issues When implementing a time domain approach, the wavefield produced by the forward step has to be available while resolving the backward step. For real cases, the data generated needs to be stored on disk and results in I/O accesses which can constitute a bottleneck. In order to reduce the amount of data stored during the forward propagation, checkpointing methods can be applied as described in Dussaud et al., 2008 for example. We will give more details on these methods in the last chapter 6.

In the case of frequency domain implementations, we avoid this I/O issues since each frequency can be treated separately. For this approach, we need instead an efficient solver. The choice of the time domain or the frequency domain depends on the problem we are solving.

Efficiency of the numerical scheme The choice of the numerical scheme is also important. Their efficiency and the quality of the image they produce depend closely of the nature and the characteristics of the subsurface imaged. Chapter 2 gives in details the advantages and limits of each numerical method.

CHAPTER 2

Seismic Modeling

This chapter is an overview of the numerical methods used for seismic modeling. This introduction to these methods is important in order to understand the algorithms and challenges faced during the implementation. This understanding offers also the opportunity to pick the appropriate method according to the underlying architectures we are studying. Usually, we have a specific numerical method widely used in a discipline. In seismic imaging, the choice of the method depends on the aims of the study, the available resources and the environment. This resulted in a multitude of methods each has its advantages and its drawbacks. In this chapter, we begin with defining to the seismic waves and describe the difference between them. Then, we introduce the main families of numerical methods approximating the equations governing wave propagation in elastic and acoustic media : integral methods, asymptotic methods and direct methods. These methods were widely studied for seismic imaging and one can find more details in [Carcione et al., 2002; Virieux et al., 2011] for example. In the last section, we give more details on finite-difference techniques since we implement them in our applications. We will go through dispersion and convergence studies. Finally, we will introduce the boundary condition methods such as the Absorbing Boundary Conditions (ABCs) and Perfectly Matched Layer (PML) methods since we approximate the wave equation in a limited domain and we need to eliminate the artifacts due to reflections on boundaries.

2.1 Seismic Waves

2.1.1 Body Waves

We distinguish two types of body waves. Compressional waves also known as the P-wave since they correspond to the first event recorded when an earthquake occurs. Then we have shear wave also called S-wave and they are recorded in second place in earthquake. Figure 2.1 illustrates the propagation directions of P- and S- waves. The S waves can be decomposed into two components: SV vertical component and SH horizontal component.

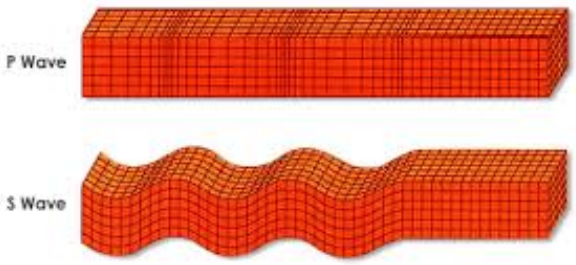


Figure 2.1 Body waves: P and S.

In a homogeneous and isotropic media the velocities of P and S waves are the following:

$$V_p = \sqrt{\frac{\lambda + 2\,\mu}{\rho}} \qquad V_s = \sqrt{\frac{\mu}{\rho}} \tag{2.1}$$

Where λ and μ are the coefficients of Lamé and ρ is the density. As the shear modulus μ is equal to 0 in fluid, S waves don't propagate in water. Also, given the expressions of velocities V_p and V_s, we notice that the P waves have always greater velocity than S waves.

2.1.2 Surface Waves

We have two types of surface waves: Rayleigh waves and Love waves. Rayleigh waves, also called ground roll, propagate when the surface of the medium is free or considered as free like air since the elastic constants and density are very low compared to their values in rocks. Rayleigh waves are dissipative this means that their amplitude diminishes with depth. Love waves, also known as Q waves, apply horizontal transformations to the surface of the earth.

Figure 2.2 depicts the behavior of the Rayleigh and Love waves in comparison to the propagation direction of the seismic wave.

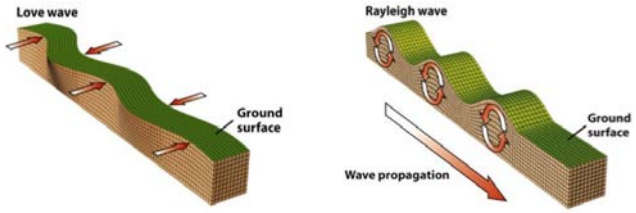


Figure 2.2 Surface waves: Rayleigh and Love.

2.2 Wave Equations

In Virieux et al., 2011 and Carcione et al., 2002, we find overviews of the different approaches to resolve wave equation depending on the needs and the application domain. We can adopt a time-domain approach or a frequency-domain approach depending on the computational needs and on the available resources.

Time Domain Approach

It simulates the variation in time of the wavefield u from sources to receivers. A usual work flow in a time domain approach consists of 3 majors steps. We first introduce a source term to create a wavefield. We then advance in time using the numerical approach chosen. When the wavefield has propagated from all sources to all the receivers used during the acquisition, we stop the computation.

The numerical methods commonly used for time-domain approaches are finite-difference and finite-element methods. They are popular for their efficiency and their straightforward implementation.

Frequency Domain Approach

This approach expresses the seismic modeling problem in the frequency domain using a Fourier transformation in time. As a result, the wave equation becomes a set of linear systems, one for each frequency. Also, the time-dependency is eliminated which permits the resolution of each systems independently.

In order to solve these systems, we can use direct solvers or iterative solvers. For direct solvers, LU and incomplete LU can be used but can cause prohibitive storage requirements in 3D cases. For the iterative solvers, we can opt for the multigrid-preconditioned Krylov methods. The main difficulty of these approaches is the convergence of the solver.

2.2.1 Elastic Wave Equation

The elastic wave equation is established using the Newton's second law of motion and Hooke's law.

Newton's law states that the acceleration of a particle when multiplied by its mass is equal to the sum of forces applied on it. We assume that we are in the case of small displacements in order to satisfy the elasticity condition.

Applying the previous laws leads to the system 2.2 which is the velocity-stress first order equation of wave propagation.

$$\begin{aligned} \rho \frac{\partial v_i}{\partial t} &= \frac{\partial \sigma_{ij}}{\partial x_j} + f_i^v \\ \frac{\partial \sigma_{ij}}{\partial t} &= c_{ijkl} \frac{\partial v_k}{\partial x_l} + f_{ij}^\sigma \end{aligned} \tag{2.2}$$

Where v_i components of velocity vector; σ_{ij} components of stress tensor; c_{ijkl} components of stiffness tensor; ρ density; f_i^v components of force source vector and f_{ij}^{σ} components of moment rate source tensor.

The displacement second-order elastodynamic equation

$$\rho \frac{\partial^2 u_i}{\partial t^2} = \frac{\partial c_{ijkl}}{\partial x_i} \frac{\partial u_k}{\partial x_l} + f_i \tag{2.3}$$

Where u_k are the components of the displacement vector.

2.2.2 Acoustic Wave Equation

Through the following sections, we use the notation $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ to denote spatial position and $t \in \mathbb{R}^+$ to denote time.

Let u denote the acoustic pressure field and ρ the density of the medium in which the acoustic wave travels. In its scalar form, the second-order partial differential equation in time can be written as follows

$$\frac{1}{\rho(\mathbf{x})c^2(\mathbf{x})} \frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - \nabla \left(\frac{1}{\rho(\mathbf{x})} \nabla u(\mathbf{x}, t) \right) = 0 \tag{2.4}$$

Where $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)^T$; ρ is the density and c is the compressional wave velocity. Density and velocity are strictly positive functions of the position $\mathbf{x}$.

2.2.2.1 Isotropic Media

An isotropic media has the same physical characteristics independently of the direction. As a consequence, the density ρ is constant and the wave equation 2.4 can be rewritten like in 2.5.

$$\frac{1}{c^2} \frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} - \Delta u(\mathbf{x}, t) = 0 \tag{2.5}$$

2.2.2.2 Anisotropic media

Transverse isotropy is the common form of anisotropy encountered in exploration campaigns. In this case, we have an axis of symmetry orthogonal to planes of isotropy. This can be the case for layered subsurfaces [Alkhalifah, 2000; Fletcher et al., 2009].

The equation 2.6 is the second-order coupled equation governing the acoustic wave propagation in a homogeneous TTI media. In this equation 2.6, we have p the pressure wavefield and q an auxiliary wavefield. We consider P and SV wave modes and denote v_{pz} the P-wave velocity in the direction normal to the symmetry plane, whereas v_{sz} designates the SV velocity normal to the symmetry plane. We also use ε and δ the anisotropic parameters. The angles θ and phi are the dip measured to the vertical and the azimuth of the axis of symmetry respectively.

$$\frac{\partial^2 p}{\partial t^2} = v_{pz}^2(1+2\varepsilon)H_2 p + \alpha v_{pz}^2 H_1 q + v_{sz}^2 H_1(p-\alpha q)$$
$$\frac{\partial^2 q}{\partial t^2} = \frac{1}{\alpha} v_{pz}^2(1+2\delta)H_2 p + \alpha v_{pz}^2 H_1 q - v_{sz}^2 H_2(\frac{1}{\alpha}p - q) \tag{2.6}$$

Where the operator H_1 and H_2 are defined as follows

$$\begin{aligned} H_1 = {} & sin^2\theta\, cos^2\phi\, \frac{\partial^2}{\partial x^2} + sin^2\theta\, sin^2\phi\, \frac{\partial^2}{\partial y^2} + cos^2\theta\, \frac{\partial^2}{\partial z^2} \\ & + sin^2\theta\, sin2\phi\, \frac{\partial^2}{\partial x \partial y} + sin2\theta\, sin\phi\, \frac{\partial^2}{\partial y \partial z} \\ & + sin2\theta\, cos\phi\, \frac{\partial^2}{\partial x \partial z} \\ H_2 = {} & \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - H_1 \end{aligned} \tag{2.7}$$

2.2.2.3 Boundary Condition : Perfectly Matched Layer

Truncating the domain where the wave propagation occurs introduce artifacts to computations. This is due to reflections of the wave on the boundaries. We add an Absorbing Boundary Condition (ABC) to blur the boundaries and thus avoid reflections.

In the finite-difference time-domain implementations we are studying, we use the Perfectly Matched Layer (PML) method as a boundary condition. This technique was designed by Jean-Pierre Berenger in 1994 for Maxwell's equation. He suggested to add an artificial layer placed adjacent to the edges of the grid as shown on the figure 2.3. The wave will be attenuated in this layer.

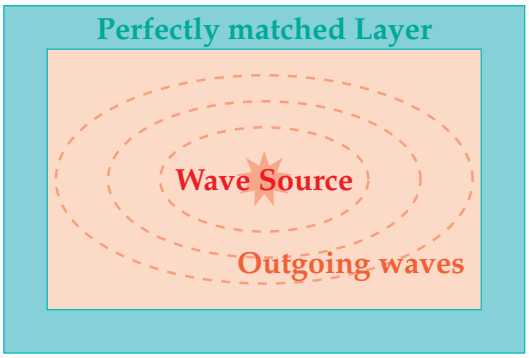
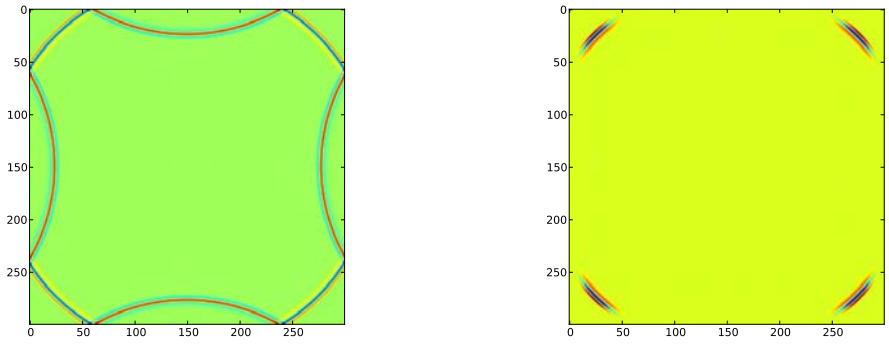


Figure 2.3 Computation Domain Truncated by a Perfectly Matched Layer.

On figure 2.4, we illustrate the impact of the PML layer on the incident wave front in an isotropic medium.



(a) Isotropic wavefield without absorbing conditions on boundaries.

(b) Absorbing conditions, PML in this example, remove the reflections on boundaries.

Figure 2.4 Impact of the PML method.

2.3 Numerical Methods for Seismic Modeling

Numerically, wave equation can be approximated using the spectral formulation or a strong formulation or a weak formulation.

Spectral formulation produces efficient results for simple geological structures whereas the strong formulation via finite-difference methods can give a good compromise between the quality of images and the computational cost. On the other hand, weak formulation via finite-elements e.g. continuous or discontinuous Galerkin methods are more suitable for areas with complex subsurfaces.

For our study, we consider a time-domain approach and approximate the wave equation using finite-difference. We will also consider two formulations of the equation. One in isotropic media and one in tilted transverse isotropic media (TTI).

2.3.1 Integral methods

These methods are based on Huygens principle that stipulates that every point in the wavefield can be considered as a secondary source.

For the integral form of the scalar wave equation in homogeneous media we use the Green function G

$$G(\mathbf{x}, \mathbf{x_s}, t) = \frac{\delta(t - |\mathbf{x} - \mathbf{x_s}|/c_0)}{4\pi|\mathbf{x} - \mathbf{x_s}|} \tag{2.8}$$

$$p(\mathbf{x}, t) = \int G(\mathbf{x}, \mathbf{x_s}, t - t')\, q(\mathbf{x_s}, t')\, d\mathbf{x_s}\, dt' \tag{2.9}$$

Green function are used as a response to a source in the studied media. The source location is $\mathbf{x_s}$. p is the pressure generated by the particles displacement in media.

These approaches are more efficient in homogeneous medium.

2.3.2 Asymptotic methods

They are also called ray-tracing methods and are used when the medium is heterogeneous. In such media, the Green's functions cannot be computed simply.

An example of the asymptotic approch is the Kirchhof approximation widely used in migration as described in Biondi, 2006.

Kirchhof approximations are based on the assumption of high frequencies.

2.3.3 Direct methods

Direct methods are based on a discretization of the computational domain. The approximation of wave equation can be done using strong formulations such as finite-difference and pseudo-spectral approaches. We can also rely on weak formulations like finite-element and finite-volume methods.

We also need a time integration in order to approximate the wave equation. Depending on the formulation chosen for the equation, the space and time derivatives can be either second or first order.

The source term is added to the right hand part of the equation in order to be able to resolve the equation.

2.3.3.1 Pseudo-Spectral Methods

Pseudo-spectral (SP) methods also known as the Fourier methods are strong formulations of partial differential equations. Using these approaches, pressure values $p(\mathbf{x})$ are approximated using basis functions ψ_j like in equation 2.10

$$p(\mathbf{x}) = \sum_{j=1}^{N} p(\mathbf{x_j})\psi_j(\mathbf{x}) \tag{2.10}$$

In the case of regular grids, one can use Fourier polynomials as basis functions. On the other hand Chebychev polynomials are used for irregular grids. In Kosloff et al., 1982, we have a description of the Fourier methods applied to forward modeling with comparison with finite-difference and finite-element methods. Contrary to finite-difference, pseudo-spectral methods are global. Modifications when they occur affect the whole computing grid. When we opt for the pseudo-spectral methods, we reduce the number of unknowns. We also reduce the number of grid points compared to finite-difference while achieving the same accuracy.

Pseudo-spectral methods can show some limitations when the topography is complex. Finer grid discretization in order to adapt to the complexity of the surface results in higher computational cost. This impacts the efficiency of this numerical method. Restraining it to relatively simple topographies.

2.3.3.2 Finite-Difference Methods

Finite-difference (FD) approaches are also a strong formulation of partial differential equations. They are based on the discretization of the computation grid

where we compute values of the wavefield. For finite-difference, we have approximations of spatial derivatives using for example the equation 2.11 where Δx is the spacing between two values of the field u.

$$\frac{du}{dx} = \lim_{\Delta x \to 0} \frac{u(x + \Delta x) - u(x)}{\Delta x} \tag{2.11}$$

The derivative can be approximated by the value given in equation 2.12. This is a forward difference approximation.

$$\frac{du}{dx} \approx \frac{u(x + \Delta x) - u(x)}{\Delta x} \tag{2.12}$$

One can have a backward difference approximation of the derivative like described in equation 2.13

$$\frac{du}{dx} \approx \frac{u(x) - u(x - \Delta x)}{\Delta x} \tag{2.13}$$

Combining the forward 2.12 and backward 2.13 approximations one can have a central approximation given by the equation 2.14.

$$\frac{\partial u}{\partial x} = \frac{u(x + \Delta x) - u(x - \Delta x)}{2\ \Delta x} \tag{2.14}$$

For higher order approximations, Taylor expansion enables the computation of the derivatives as well as an estimation of the error.

In Moczo et al., 2007 and Kelly et al., 1976, we have detailed studies of the finite-difference methods applied in isotropic and anisotropic medium. All aspects ranging from the grids used to the boundary conditions are discussed.

Major difficulties in FD methods are due to the discretization grids. The space steps are constrained by the minimal value of the velocity in the media. For heterogeneous medium, the space discretization requires lots of computations.

FD methods are inefficient when the topography is complex. In order to have a precise approximation, space discretization steps need to be very small which results in a huge computational demand.

Finite-difference methods can be applied in both frequency and time domains. Frequency domain may be more efficient in inversion problems than time domain when multiple source locations are used [Pratt, 1999; Virieux et al., 2009; Operto et al., 2009]. In the case of forward modeling, time domain is widely used since it is more adapted to the computation requirements of such applications [Moczo et al., 2007; Virieux et al., 2011].

Finite-difference methods need to satisfy important conditions in order to guaranty their effectiveness. It consists of stability, convergence and consistency :

- *Stability* means that the solution is bounded when the analytical solution of the PDE is bounded.
- *Consistency* means that the truncation tends to zero when the spatial grid spacing and the time tend to zero.

- *Convergence* is satisfied when the solution using a finite-difference approach the analytical solution of the partial differential equation.

For finite-difference methods, we can have either an explicit or an implicit scheme. Explicit schemes use previous values of the wavefield in order to update a given grid point for the current time step. On the other hand, implicit scheme updates the whole grid at the current time step using an inversion of a matrice [Chu et al., 2012].

2.3.3.3 Finite-Element Methods

Finite-element (FE) methods are weak formulation of partial differential equations. We use for these methods functions as basis to approximate the wavefield.

Finite-element approaches are more efficient in complex topography than finite-difference and pseudo-spectral methods. They enable a more precise approximation of the surface.

According to the conditions we can ensure in approximating wave motion, we have different ways to adopt the finite-element method. The following paragraphs give more details on the different variations of this method.

Continuous Finite-Element

Continuous finite-element methods such as spectral-element (SE) suppose that the wavefield is continuous on the boundaries separating the elements where we compute locally its values or gives explicitly the values of the wavefield at the boundaries.

Discontinuous Finite-Element

Discontinuous Galerkin methods remedy to the strong affirmation we have in the continuous formulation of the standard finite-element methods. These new methods introduce the notion of flux exchange instead.

2.4 Application to the Acoustic Wave Equation

In this section, we detail the necessary steps for the application of a finite-difference time domain (FDTD) to acoustic wave equations. We consider isotropic and anisotropic media and provide the formulas in each case.

2.4.1 Isotropic Media

We assume constant density and velocity in the media studied. We consider the wave equation 2.5 and the FDTD approach as mentioned above. The discretization is performed on both space and time domains enabling the computation of the values of the wavefield.

Let us denote $\mathbf{x}_{ijk} = (i\Delta x, j\Delta y, k\Delta z)$ and $t^n = n\Delta t$ where $\Delta t, \Delta x, \Delta y$ and Δz designate time and space discretization steps respectively and $\{i, j, k, n\} \in \mathbb{N}$. u^n_{ijk} is the solution computed for the point $\mathbf{x}_{ijk}$ for the time step t_n.

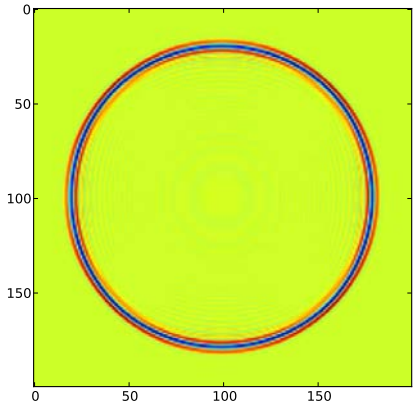


Figure 2.5 Wavefront in isotropic media.

2.4.1.1 Time Discretization

The numerical scheme used for time discretization is explicit and centered. Our goal is to compute a second order accurate approximation of $\frac{\partial^2 u^n_{ijk}}{\partial t^2}$.

The standard approach to evaluating this approximation is to expand each of the function values of u in a Taylor series about the point t:

$$\begin{aligned} u(\mathbf{x}, t + \Delta t) &= u(\mathbf{x}, t) + \Delta t \, \frac{\partial u(\mathbf{x}, t)}{\partial t} + \frac{\Delta t^2}{2} \, \frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} + O(\Delta t^3) \\ u(\mathbf{x}, t - \Delta t) &= u(\mathbf{x}, t) - \Delta t \, \frac{\partial u(\mathbf{x}, t)}{\partial t} + \frac{\Delta t^2}{2} \, \frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} + O(\Delta t^3) \end{aligned} \tag{2.15}$$

Summing the above expressions leads us to the expression of the approximation :

$$\frac{\partial^2 u(\mathbf{x}, t)}{\partial t^2} = \frac{u(\mathbf{x}, t + \Delta t) - 2u(\mathbf{x}, t) + u(\mathbf{x}, t - \Delta t)}{\Delta t^2} \tag{2.16}$$

Thus the approximation of the time derivative for $(\mathbf{x}_{ijk}, t^n)$ is :

$$\frac{\partial^2 u^n_{ijk}}{\partial t^2} = \frac{u^{n+1}_{ijk} - 2u^n_{ijk} + u^{n-1}_{ijk}}{\Delta t^2} \tag{2.17}$$

2.4.1.2 Space Discretization

Similarly to the time discretization, we opt for an explicit centered scheme. The approximation we are intending to compute is p order accurate. Hence, we will need $p + 1$ points of the grid to compute this approximation.

We proceed as in the previous paragraph and we write Taylor series in order p for each dimension of the domain. We obtain the following approximation of the Laplace operator :

$$\Delta u_{ijk}^n = \frac{1}{\Delta x^2}\sum_{l=-p}^{p} a_l u_{i+l,j,k}^t + \frac{1}{\Delta y^2}\sum_{l=-p}^{p} a_l u_{i,j+l,k}^t + \frac{1}{\Delta z^2}\sum_{l=-p}^{p} a_l u_{i,j,k+l}^t \tag{2.18}$$

When $l \in \{-\frac{p}{2}, \frac{p}{2}\}$, the constants a_l are the Taylor coefficients for the order p. The values of these coefficients are shown in table 2.1 for different accuracy orders. Since the numerical scheme we are using is centered, the coefficients a_l and a_{-l} are the same.

Order	$\mathbf{a_0}$	$\mathbf{a_1}$	$\mathbf{a_2}$	$\mathbf{a_3}$	$\mathbf{a_4}$	$\mathbf{a_5}$	$\mathbf{a_6}$	$\mathbf{a_7}$
2	-2	1						
4	$\frac{-5}{2}$	$\frac{4}{3}$	$\frac{-1}{12}$					
6	$\frac{-49}{18}$	$\frac{3}{2}$	$\frac{-3}{20}$	$\frac{1}{90}$				
8	$\frac{-205}{72}$	$\frac{8}{5}$	$\frac{-1}{5}$	$\frac{8}{315}$	$\frac{-1}{560}$			

Table 2.1 Taylor Coefficients

Combining all the previous results, 2.18 and 2.17, in equation 2.5 gives us the final formula 2.19 where the wavefield is calculated using a discretization of order $2p$ in space and 2 in time.

$$\begin{aligned} u_{i,j,k}^{t+1} =& 2u_{i,j,k}^t - u_{i,j,k}^{t-1} + c^2\Delta t^2 \left\{ \frac{1}{\Delta x^2}\sum_{l=-p}^{p} a_l u_{i+l,j,k}^t \right. \\ &\left. + \frac{1}{\Delta y^2}\sum_{l=-p}^{p} a_l u_{i,j+l,k}^t + \frac{1}{\Delta z^2}\sum_{l=-p}^{p} a_l u_{i,j,k+l}^t \right\} \end{aligned} \tag{2.19}$$

The finite-difference time-domain applications we are going to study in the next chapters are based on the formula 2.19. We will give further explanations when describing their algorithms.

2.4.2 Anisotropic media

In the case of anisotropic medium, we use the same discretization techniques in time and space applied to the equation 2.6.

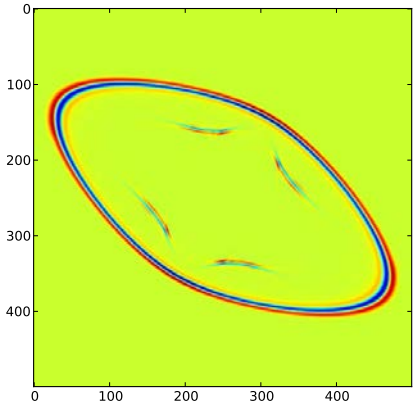


Figure 2.6 Wavefront in anisotropic TTI subsurface where $\alpha = 1$, $\theta = \frac{\pi}{6}$, $\phi = \frac{\pi}{3}$, $\varepsilon = 0.24$ and $\delta = 0.1$.

2.4.3 Stability Condition and Dispersion

When solving hyperbolic partial differential equations numerically, we need a condition for convergence. We consider the CFL condition named after Richard Courant, Kurt Friedrichs and Hans Lewy who described it in their paper [Courant et al., 1967].

For wave equation, the time step Δt must be less than the time for the wave to travel adjacent grid points. Thus, when the grid point separation (Δx, Δy and Δz) is reduced, the upper limit for the time step also decreases.

For a general n-dimensional problem and in the case of a p-order approximation in space, the CFL condition is given by equation 2.20.

$$c\,\Delta t \sum_{i=1}^{n_{dim}} \frac{1}{\Delta x_i^2} \leq 2\left[n_{dim} \sum_{l=-\frac{p}{2}}^{\frac{p}{2}} |a_l|\right]^{-1/2} \tag{2.20}$$

CHAPTER 3

Performance Study of FDTD Applications

Among the numerical methods used in seismic modeling to approximate the wave equation as described in chapter 2, we focus on the finite-difference time-domain (FDTD) approach since it is widely implemented in seismic imaging applications. It represents the computational part of these applications and contributes greatly to their execution time. For example, Ortigosa et al., 2008 report that in an implementation of the Reverse Time Migration, 90% of the execution time is spent in the computational part based on finite-difference approach. In this chapter, we extract an FDTD proxy and focus on the study of its performance. The aim is to deeply understand the behavior of this proxy at node level in order to highlight the architectural features impacting their performance. This characterization is useful for the co-design as it permits the extrapolation of the FDTD performance on future architectures. Possible optimizations on the core and node levels are studied. We detail their advantages and their limits. We position all these study in the context of a co-design process.

We will begin with a presentation of the performance study levels as they are used for the co-design. We explore state-of-the-art performance modeling techniques. Then, we give a deep insight on performance of isotropic and titled transverse isotropic (TTI) FDTD kernels. These implementations are the building blocks of the Reverse Time Migration (RTM). The whole application will be studied in the last chapter 6.

3.1 Overview of Performance Modeling Techniques

Performance characterization can be split into 5 categories. These levels correspond to a different stage in performance study process. The last 4 levels tightly related to the underlying hardware and correspond to different levels of the architecture. The figure 3.1 is a high level summary of the different stages of performance study as a preparation for the co-design process.

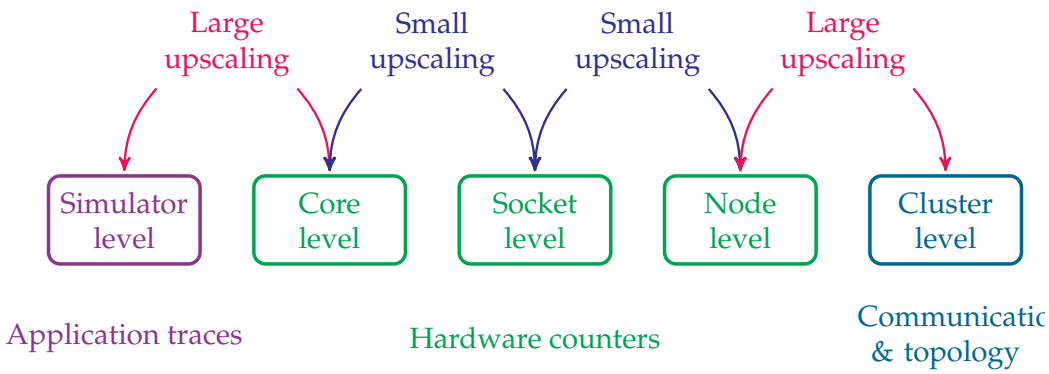


Figure 3.1 From Imbert et al., 2011. Incremental performance study levels .

Simulators Cycle accurate CPU simulators are very interesting when the underlying architecture is not accessible. Simulators need application traces as an input in order to generate the simulated performance. This gives us a starting point to identify the bottlenecks on future architectures.

Core Level On this stage, the performance study aims to find out the most important optimizations such as vectorization, prefetching, loop modifications, cache utilization, etc.

Socket Level We consider at this level multi-threading and shared memory parallelization. It's important to see their effect on bandwidth. It also means a better management of the shared caches and the reuse of data fetched into the last level cache.

Node Level Here, we are more concerned by the Non Uniform Memory Accesses (NUMA) between sockets. Threads pinning is of major importance in order to remove performance drop due threads migration from one socket to another. A well known technique to prevent NUMA effect is the first-touch that permits to allocate data the nearest possible to the thread that made the allocation.

Cluster Level We use distributed memory configurations and introduce another programming models to handle communications such MPI. Interconnect latency can be penalizing in some networks which can be the cause of poor performance in applications where we perform a lot of communications and I/O movements. We also need to have load balancing approach mainly in heterogeneous clusters and for applications where tasks spread among nodes are not similar and some of them are likely to take more time to finish.

The Advanced Scientific Computing Research (ASCR) report on performance modeling and simulation [Hoisie et al., 2012] defines the different approaches to model performance of scientific applications. We present 3 different techniques: analytical models, trace-based models and Roofline models.

3.1.1 Analytical Models

In analytical or semi-analytical models, we capture the performance behavior in mathematical formulas where machine parameters such as number of cores, bandwidth, cache size, are considered as knowns. We can also use statistics to elaborate these formulas. In some cases, this method depends on the underlying implementation. On the other hand, compilers and operating systems influence is usually discarded by this type of modeling which results in discrepancies between the actual performance and the prediction. For a fixed implementation, the models can be extrapolated to other architectures which represent an interesting task to initiate before porting to new architectures. Models with few relevant hardware parameters can be extremely beneficial to predict performance.

Analytical models can be very helpful for auto-tuning. For some optimization techniques, using the hardware features as an input enables to compute necessary parameters of the optimizations while maximizing the performance formulas.

3.1.2 Trace-based Models

Traces and measurements collected using performance tools can also help to characterize performance. We can think of communications patterns for example. In the case of the applications parallelized using the message passing approach, traces can visualize compute time and communication time per worker. Possible load imbalance between the workers can easily detected. These traces can concern a single event such as DRAM bandwidth or the application runtime.

3.1.3 Roofline Models

The Roofline model, as introduced by Williams et al., 2009, is a graph gathering useful informations on the architecture studied and gives an insight on its impact on performance of a given application.

On a Roofline model, we plot the peak performance values of the underlying machine. It consists of the floating point peak performance and the peak bandwidth. One can use the sustained bandwidth measured with a benchmark like STREAM. The floating point performance can also be adjusted to the theoretical performance of the algorithm under consideration. These new values give a more realistic estimation of the achievable performance on the machine used.

The figure 3.2 is an illustration of the Roofline model. On the x-axis, we have the arithmetic intensity which is the ratio of the number of the floating point operations by the number of bytes transferred between the processing unit and memory. On the y-axis, we have the performance in terms of Gflop/s. Performance measurements are done using hardware counters for example.

Depending on which side of the arithmetic intensity of the machine the performance is located, one can determine whether the application is bandwidth or compute bound. We can also have a preliminary estimation of the efficiency of the algorithm compared to the peak performances of the machine. For example, kernel 2 is less efficient than kernel 3. As a result, kernel 2 is subject to optimizations in order to perform better on the computing resources.

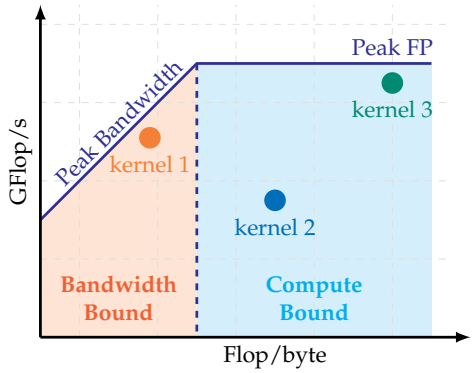
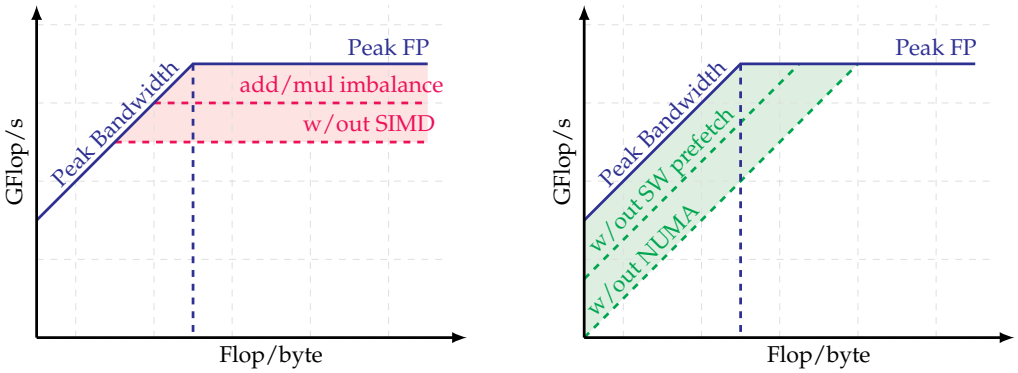


Figure 3.2 Example of the Roofline model. Kernel 1 is limited by the bandwidth available on the machine. On the contrary, kernel 2 and kernel 3 are limited by the compute capabilities of the hardware.

On figure 3.3, we place schematically the performance gain when optimizations are applied gradually on both sides. In order to reduce the gap between the actual performance and the estimation of peak performances, we proceed gradually with these optimizations.



(a) Core optimizations.

(b) Bandwidth optimizations.

Figure 3.3 Optimizations on the Roofline model.

3.2 Performance Modeling of FDTD

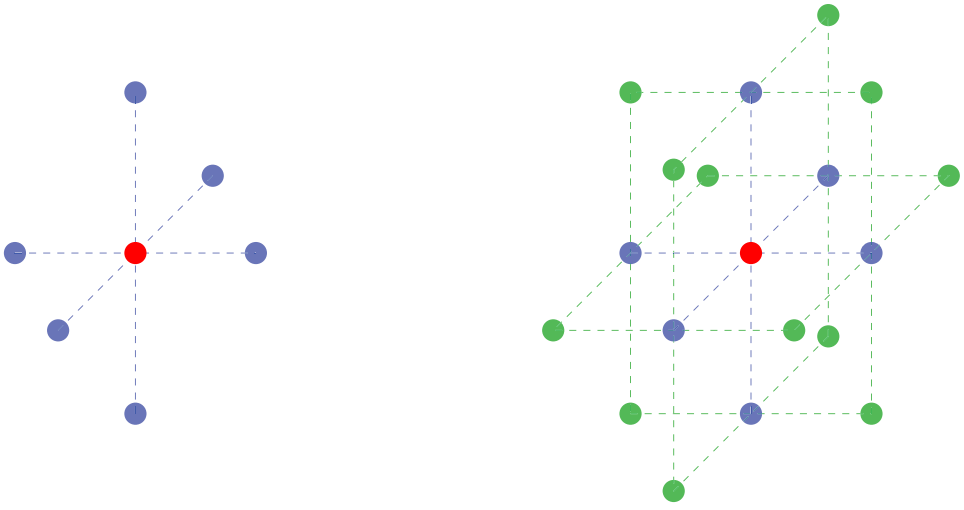
In this section, we present the algorithm that results of the equations described in chapter 2. We also present a performance model for theoretical performance. For this aim, we make some assumptions on the cache behavior and on the number of floating point operations performed.

3.2.1 FDTD Algorithm

The finite-difference techniques applied to the equations 2.5 and 2.6 conducted us to discrete formulations approximating the pressure field on the compute grid. For example, we obtain the equation 2.19 in the isotropic medium. Contributions of neighboring values at a time step t are required to update the value of a central point at time step $t + 1$. The pattern visualizing these contributions in order to compute a central value is called stencil.

The order of the stencil corresponds to the order of the Taylor expansion used to approximate the partial differential derivatives. For example, the stencil describing the Laplacian in equation 2.18 is of order $2.p$. Discretization in time is of order 2.

Depending on the medium where the wave equation is approximated, we may have different patterns. For example, the 7-point stencil in figure 3.4a can be used in an approximation in isotropic medium using an order 2 in space. On the other hand, the figure 3.4b represents a stencil used in the case of tilted transverse isotropy (TTI). The second-order mixed derivatives in the equation 2.6 implies the use of the values of the diagonal neighbors.



(a) Isotropic stencil when $p = 1$ **(b)** Stencil of TTI when $p = 1$

Figure 3.4 Centered grid.

The algorithm 1 is the result of the formula 2.19. Some implementations perform the update of the array u in place reducing the memory required to save the intermediate values of the wavefield. In our implementation, we opt for double buffering and use two arrays u_{t-1} for time steps $t - 1$ and u_t for t. The values for $t + 1$ are stored in u_{t+1}. The roles of u_{t-1} and u_t are inverted. This method is likely to facilitate the parallelization of the code even if it implies more data to allocate in memory.

3.2.2 Modeling Theoretical Peak Performance

We now consider the peak performance of the FDTD applications. The theoretical peak performance can be computed using the number of floating point operations and the number of bytes in the algorithm 1 for example. For this purpose, we need to make some assumptions regarding the architecture. The table 3.1 summarizes the number of the floating point operations of our isotropic and TTI codes per a point update when assuming an infinite cache and an infinite memory bandwidth. We also give the number of DRAM reads and DRAM writes.

Algorithm 1 4D loop to compute the wavefield u in an isotropic medium.

for $t \leftarrow Tmin, Tmax$ **do**
  for $k \leftarrow 1, N_z$ **do**
    for $j \leftarrow 1, N_y$ **do**
      for $i \leftarrow 1, N_x$ **do**

$$
\begin{aligned}
Laplacian\_u = {} & a_0 * u_t(i,j,k) \\
& + a_1 * \{u_t(i-1,j,k) + u_t(i+1,j,k) \\
& \quad + u_t(i,j-1,k) + u_t(i,j+1,k) \\
& \quad + u_t(i,j,k-1) + u_t(i,j,k+1)\} \\
& \ldots \\
& + a_p * \{u_t(i-p,j,k) + u_t(i+p,j,k) \\
& \quad + u_t(i,j-p,k) + u_t(i,j+p,k) \\
& \quad + u_t(i,j,k-p) + u_t(i,j,k+p)\}
\end{aligned}
$$

$$u_{t+1} = 2\,u_t - u_{t-1} + c^2\,\Delta t^2\,Laplacian\_u$$

      end for
    end for
  end for
end for

	Add	Mul	Flops/update	DRAM read	DRAM write
Isotropic	$6p+2$	$p+3$	$7p+5$	3	1
TTI	$30p+38$	$18p+62$	$48p+100$	24	2

Table 3.1 Theoretical Peak Performance

Microarchitecture	**Intel Westmere EP Xeon X5680**
Clock[GHz]	3.33
Node sockets/cores/threads	2 / 12 / 24
FP peak performance [GFlop/s]	159.84
L1/L2/L3 cache	32KB / 256KB / 12MB
Socket theoretical bandwidth [GB/s]	32.0
Socket sustained bandwidth [GB/s]	21.0

Table 3.2 Test machine specifications.

Let us consider the machine described in 3.2 and the isotropic kernel. If we maintain the hypotheses of infinite cache and infinite memory bandwidth, a theoretical peak performance of this FDTD kernel can be computed using the total number of floating point operations nFP and is equal to $Peak_1$ given by equation 3.1.

$$Peak_1 = \frac{nFP}{8\ Flop/cycle} \tag{3.1}$$

On the Westmere machine 3.2, independent adds and multiplications can be performed in parallel which results in a second estimation of the theoretical peak performance equal to $Peak_2$ defined in equation 3.2.

$$Peak_2 = \frac{max(nAdd,\ nMul)}{4\ Flop/cycle} \tag{3.2}$$

As a result, the efficiency of the peak performance for the isotropic kernel is equal to the expression 3.3 and illustrated in figure 3.5 as a function of the half stencil order.

$$\begin{aligned} Efficiency &= 100 \times \frac{Peak_1}{Peak_2} \\ &= 50 \times \frac{7p+5}{6p+2} \end{aligned} \tag{3.3}$$

The same discussion can be applied to the anisotropic kernel TTI in order to compute the efficiency using the theoretical values of floating point operations and DRAM read and write in table 3.1

In the first estimation of the peak performance, we considered an infinite memory bandwidth and an infinite cache. The performance is only limited by the compute resources available in the underlying architecture. However, this value is never reached in real cases since memory bandwidth and caches will be limiting factors. As a consequence, we can compute an other estimation for the performance given the theoretical memory bandwidth β. We will also need a characterization of the algorithm. We use the arithmetic intensity defined as the ratio of the theoretical floating point operations nFP by the number of the bytes transferred

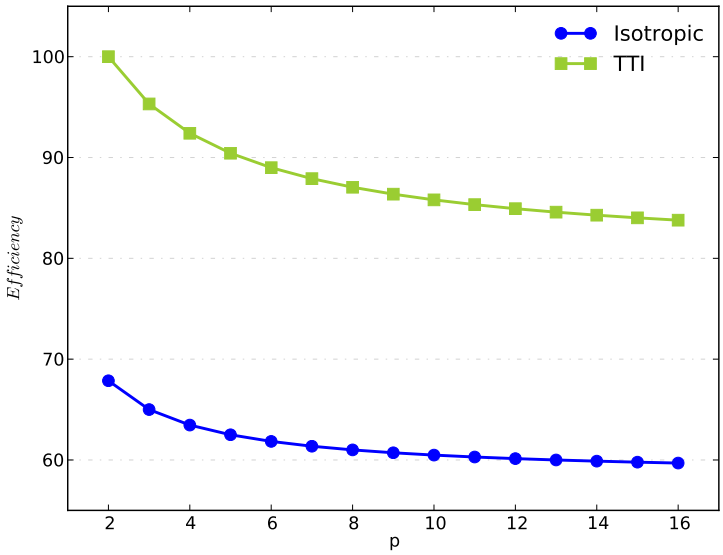


Figure 3.5 Efficiency of the peak performance for isotropic and TTI kernels when the cache is infinite.

nB. For this new estimation, we will maintain the assumption of an infinite last level cache. The expression 3.4 gives the definition of this new estimation and its value in the isotropic case.

$$\begin{aligned} Peak_{algo} &= \beta \times \frac{nFP}{nB} \\ &= \frac{\beta}{sizeofreal} \times \frac{7p+5}{3+1} \end{aligned} \tag{3.4}$$

The efficiency of this new theoretical estimation compared to the machine floating point peak performance is given by the expression 3.5. We consider the Westmere machine in 3.2. Its clock frequency is equal to 3.33 GHz and its theoretical bandwidth per socket is equal to 32 GByte/s. The figure 3.6 illustrates the results for isotropic and TTI applications (only the single precision case is considered).

$$\begin{aligned} Efficiency &= \frac{Peak_{arch}}{Peak_{algo}} \\ &= \frac{8 \times freq}{\frac{\beta}{4} \times \frac{7p+5}{3+1}} \end{aligned} \tag{3.5}$$

We could also consider the maximum sustainable bandwidth measured with the STREAM benchmark [McCalpin, 2000] for example. The new upper limit on the peak performance will decrease and consequently will be a more accurate value compared to the previous estimations.

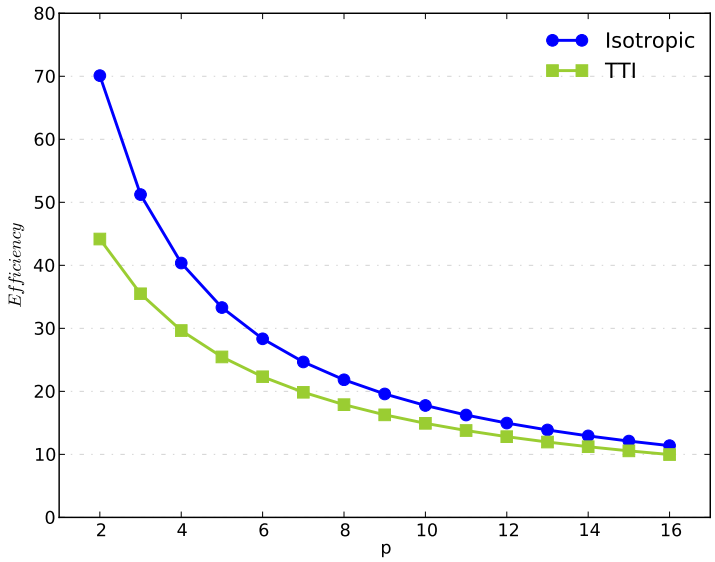


Figure 3.6 Efficiency of the peak performance estimation given the memory bandwidth in comparison with the floating point peak performance of the machine. We consider the isotropic and TTI implementations.

The estimation of the peak performance on a given machine is an important step in studying the performance of applications. This is the first step we need to perform before starting to think about the possible optimizations of the code. We are going to use these values in the following sections in order to identify methodically the possible optimizations that can be applied on FDTD applications.

3.3 Performance Optimizations of FDTD

In this section, we present state-of-the-art optimizations performed on stencil computations on node level. We explain the advantages and limitations of each strategy. We also consider a multi-threaded version of the FDTD applications in order to fully benefit from the resources available.

For the node level, we are going to overview mechanisms such as the software prefetching in modern architectures and its impact on performance. We will also study the NUMA effect in multi-socket configurations and highlight the solutions to get around this issue. Then, we consider cache optimizations to cope with memory bandwidth and latency limitations. The goal of these optimizations is to enhance data locality in caches. Cache optimizations can be hardware-dependent such as spatial and temporal blocking. They can also hardware-free and use a cache oblivious approach. As a conclusion to this paragraph, we explore the auto-tuning techniques applied to stencil codes as they offer the opportunity to quickly produce an optimized code for an underlying architecture.

In a multi-node level, the focus is rather held on communications and load bal-

ancing between processes. This implies domain decomposition strategies, communications overlapping with computations, work stealing and scheduling.

3.3.1 NUMA-Awareness

In the case of multi-core architectures with more than a single socket per node, we can be confronted to the NUMA effect. This effect is noticed for the codes using multi-threading via mechanisms offered by an appropriate programming model like openMP. The effective allocation of the arrays and data used by threads occurs when they are accessed the first time and in some cases the allocation can take place in the memory of the neighboring socket, resulting in asymmetric memory accesses and thus additional latencies which imply dramatic drop in performance. One can remedy to this issue using the *first touch* mechanism. It consists of accessing the arrays before starting the computation using the same pattern in order to pin them in the appropriate memory bank.

3.3.2 Prefetching

Prefetching is a mechanism that reduces the pressure on the memory bandwidth. It anticipates the upcoming requests of the program and brings into caches more elements than initially needed. The following requests can be satisfied quickly since the required element is already in the cache. Prefetching can be implemented in two ways: in the hardware or in the software [Lee et al., 2012].

3.3.3 Cache Optimizations

Cache optimizations aim to use efficiently the cache hierarchy in order to hide memory latencies and reduce the pressure on the bandwidth. This is possible when the locality principle is applied properly. For caches, it consists in the spatial locality and the temporal locality.

- *Spatial locality* When an element is fetched to the cache upon a DRAM read request, the other elements on the same cache line will be used.
- *Temporal locality* Data already in the cache can be reused for the future computations.

Within a program, a data request upon a read for example can generate misses when cache can not serve the request. Misses can have three reasons.

- *Compulsory misses* occur when the program starts. Data needed by the instructions are not yet fetched in the cache. They are also called cold start misses.
- *Capacity misses* happen when all elements used by the program can not hold in the cache. This can generate high DRAM traffic in order to serve constantly the instructions requests.
- *Conflict misses* result from conflicting memory addresses that have the same cache banks.

3.3.3.1 Spatial Blocking

Spatial blocking aims to enhance data locality in order to reduce capacity misses. This optimization increases data reuse in cache and thus reduces the amount of data brought from memory. This is an important optimization approach for memory bound and latency bound applications. Codes as stencil computations reap a gain in performance from spatial blocking.

For spatial blocking, the 3D loop on the computation grid is decomposed into smaller loops that enables data to fit into the last level cache. The algorithm 1 is transformed into the algorithm 2 where we introduce loop steps $block_x$, $block_y$ and $block_z$. These are the blocking factors along the X, Y and Z axis respectively. The aim is to bring a block into the cache and to perform all computations for the time step t+1 on its elements.

The figure 3.7 is an illustration of the spatial blocking as described in algorithm 2. The green arrows show the data access pattern inside the block.

Algorithm 2 3D loop used to compute the wavefield u at time step t+1 in an isotropic medium using cache blocking.

for $k \leftarrow 1, N_z, block_z$ **do**
  for $j \leftarrow 1, N_y, block_y$ **do**
    for $i \leftarrow 1, N_x, block_x$ **do**
      $Laplacian\_u = a_0 * u_t(i,j,k) + ...$
      $u_{t+1} = 2\,u_t - u_{t-1} + c^2\,\Delta t^2\,Laplacian\_u$
    end for
  end for
end for

The effectiveness of cache blocking strategy depends of the blocking factors. Their sizes should take into consideration the size of the last level cache, the number of threads sharing the cache and the number of arrays involved in the algorithm. Determining the optimal block sizes can be done using analytical performance models [Datta et al., 2008].

In the following paragraphs, we denote C the cache size and N_{th} the number of threads sharing this cache.

3D Blocking This approaches is described in figure 3.7. The blocking is performed along the X, Y and Z dimensions. The block size is equal to $blocksize = block_x \times block_y \times block_z$ and needs to satisfy the inequality 3.6.

$$N_{th} \times block_x \times block_y \times block_z \leq C \tag{3.6}$$

In modern architectures, software prefetching is an important mechanism used to enhance spatial and temporal localities in caches and to lessen the pressure on memory bandwidth. Studies like [Datta et al., 2008; Rivera et al., 2000] showed that cache blocking can interfere with software prefetching and results in a regression of the overall performance compared to the non-blocked version. As shown on

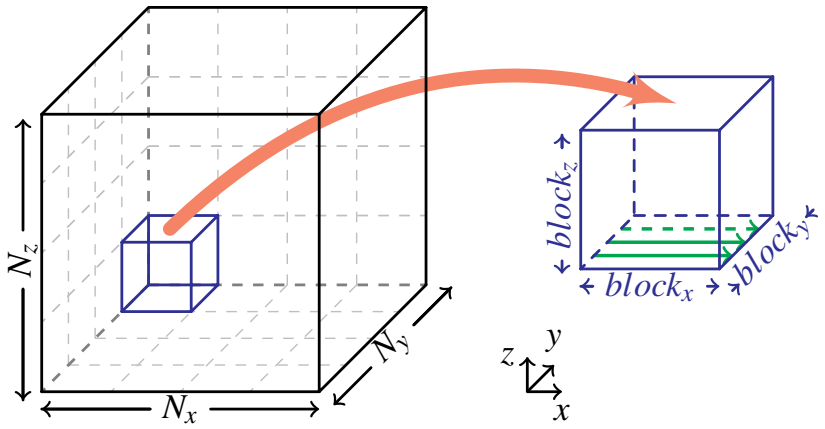


Figure 3.7 Spatial blocking of the computation grid using the blocking factors $block_x$, $block_y$ and $block_z$. The unit stride direction is on the X dimension. The green arrows inside the block indicate how the data is accessed.

figure 3.7, the unit stride access within the block can be smaller than the prefetching distance. As a consequence, prefetching will bring into the cache elements that are not useful for the computations on the current block.

2.5D Blocking This approach was introduced by Rivera et al., 2000 for a Jacobi kernel. It is also used by Nguyen et al., 2010 for their stencil codes. It performs blocking only on the X and Y dimensions and stream on the Z direction. The figure 3.8 gives an example for a stencil of order 2 in space. In order to compute the values of elements in the plane k at time step t+1, we need to bring into the cache the $k-1$, k and $k+1$ planes at time step t. In a general case, for a stencil of order $2p$ in space, we need to hold in cache only $(2p+1)$ XY planes. The size of these XY planes is equal to $block_x \times block_y$ which leads to the inequality 3.7.

$$N_{th}\ (2p+1)\ block_x\ block_y \leq C \tag{3.7}$$

The figure 3.8 is an illustration of the 2.5D blocking for a stencil of order 2 in space. The colored planes depict the elements that we need to keep in the cache in order to reuse it efficiently while updating the wave field.

Taking into consideration the possible interaction between prefetching and blocking as described in [Datta et al., 2008], we tend to maximize the length of the block on the unit stride dimension (X in this case) and choose it equal to the grid size N_x. This limits the unknowns in the previous inequality 3.7 to $block_y$.

$$block_y \leq \frac{C}{(2p+1)\ N_x\ N_{th}} \tag{3.8}$$

As a result the size of $block_y$ needs to satisfy the system of inequalities 3.9. The figure 3.9 delimits the region containing the possible values when $N_x = N_y$.

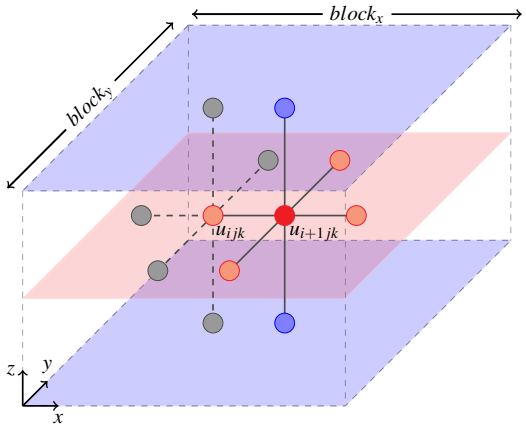


Figure 3.8 For simplicity matter, we consider a half stencil order $p = 1$. In this example we update the grid elements on the red plane for time step t+1 using elements contained in the red plane and in blue planes in time step t. For $p = 1$, we only need to keep 3 XY planes in cache each of size $block_x \times block_y$.

$$\begin{cases} block_y \leq \frac{C}{(2p+1)\ N_x\ N_{th}} \\ block_y \leq N_y \end{cases} \tag{3.9}$$

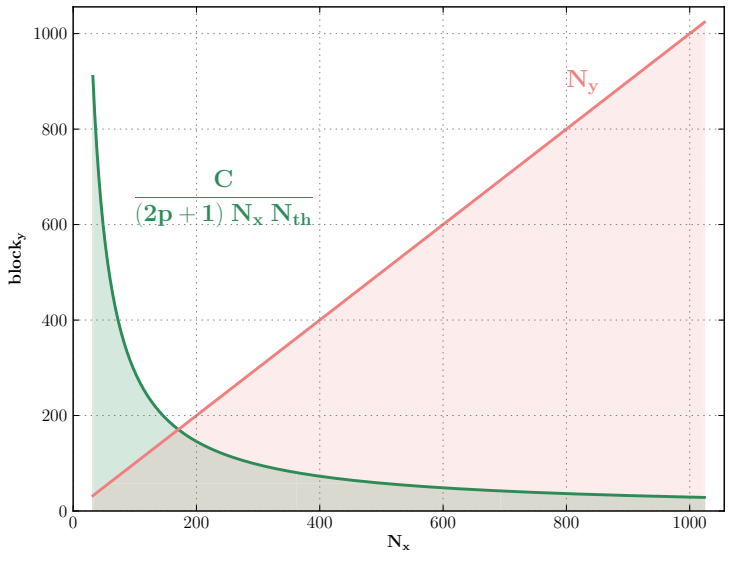


Figure 3.9 The size of the block along the y direction is contained in the region resulting of the intersection between the green and pink regions which give the values satisfying the inequalities of the system 3.9.

When blocking along the y direction and streaming along the z direction, we ensure for each thread an optimal data reuse. The update of the first red plane as

illustrated in figure 3.8 requires the fetching of the $(2p + 1)$ XY planes. Within a single time step and for the following planes, we only need to load a single plane XY since the remaining $2p$ planes are already in cache.

3.3.3.2 Temporal Blocking

Similarly to spacial blocking, the temporal blocking aims to increase the temporal locality of the data. In this case, the blocking is performed on the outermost loop in algorithm 1. We perform $block_t$ iterations on the same data before loading the next data set. This reduces the number of data transfers due to the intermediate write-backs and loads. Many methods exist for temporal blocking. We introduce the time skewing and the temporal wavefront blocking approaches.

Time Skewing. It is a loop transformation that exchanges the spacial and temporal loops. As a result, the $block_t$ time steps are executed on the same elements. As illustrated on figure 3.10 for a 3-point stencil, we perform multiple computations in the blue area and we load a single element. On the figure 3.10, we only have 1D illustration but this temporal blocking can also be applied to 2D and 3D computations.

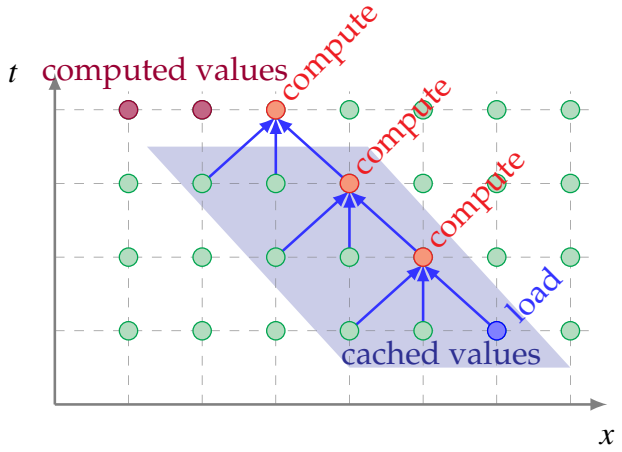


Figure 3.10 Time skewing for $p = 1$. We only keep elements in the blue area. We can perform multiple computations and only load an elements.

Temporal Wavefront Blocking As described in [Wellein et al., 2009],[Treibig et al., 2011] and [Wittmann et al., 2010], wavefront blocking takes advantage of the shared cache in the multi- and many-cores architectures. Wave updates are performed in a pipeline fashion among threads sharing a last level cache (L2 or L3). Depending on the size of the stencil considered and after the pipeline start-up by the first thread, the remaining threads update the plane for the next time steps. Synchronizations are needed though to prevent from race conditions. The aim of this blocking is to reduce the pressure on bandwidth and ensure better temporal

locality of data. Figure 3.11 illustrates the wavefront blocking approach for a stencil of order 2 ($p = 1$).

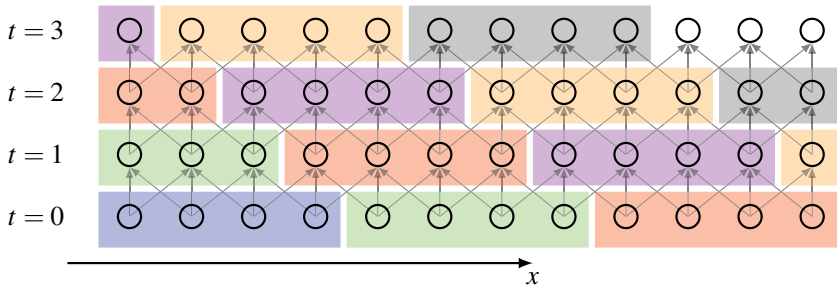


Figure 3.11 Wavefront blocking for $p = 1$ and cache group equal to 4. Elements in the block are computed by the same thread. Blocks with the same color are computed by different threads simultaneously.

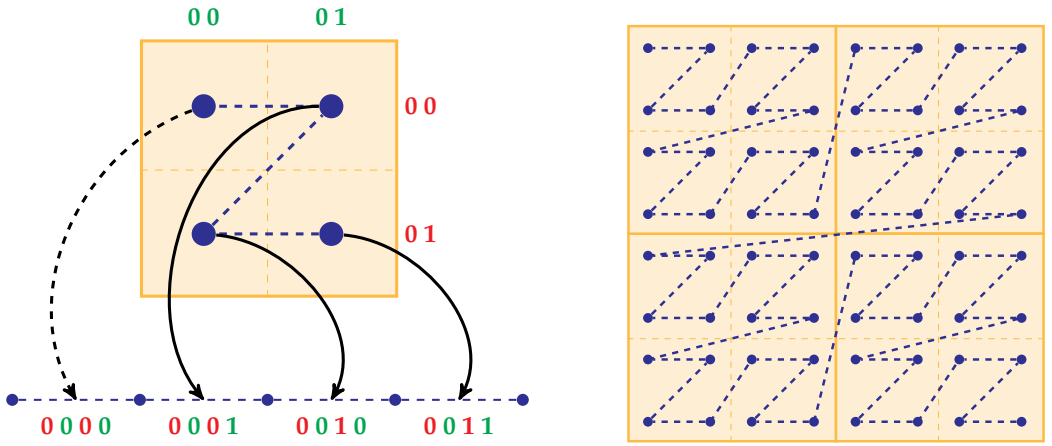
3.3.3.3 Cache Oblivious

Cache oblivious is an approach developed by Frigo et al in [Frigo et al., 1999] that overcomes the hardware features of the machine e.g. cache size and number of threads and performs an optimal blocking that uses finer grain than spatial blocking. The main idea of this algorithm is based on divide and conquer technique. As the resulting algorithm modifies greatly the initial implementation of the algorithm 1, it can be judicious to implement it in an auto-tuning framework such as PATUS [Christen et al., 2011]. Some programming models adopted the concept of cache oblivious and integrated it in the runtime in order to have straightforward use of this algorithm capabilities, namely Cilk.

3.3.4 Z-order Curve

The z-order curve, also known as Morton order, is a particular case of the space-filling curves [Butz, 1968]. The main idea of this approach is to define a function that maps coordinates of elements contained in a multidimensional space to 1D space. The curve of this function connects these elements and defines a path within the domain.

This technique is used for data management in data intensive applications to cope with latencies and bandwidth limitations when fetching data from disk or memory [Pascucci et al., 2001; Mellor-Crummey et al., 2001]. It is also used for computations reordering to enhance data locality [Jin et al., 2005; Davis et al., 2011]. This approach can be very useful for cache sensitive applications where data reuse has a great impact on performance. The native z-order curve is applied to domains with dimensions power of two. It uses binary representation of the coordinates in this first domain and combines them in order to compute the corresponding coordinate in the new 1D domain. Figure 3.12a is an example of the coordinates conversion from a 2D to 1D. Figure 3.12b gives the z-order curve when considering multiple levels of data structures. Threads, such as OpenMP threads,



(a) Coordinates conversion for the z-order curve. Interleaving the binary representations gives the coordinates in the 1D space.

(b) The z-order path recursively accessing data structures from wider to finer grain parallelism.

Figure 3.12 Illustration of the z-order approach.

share the computation grid. In this particular example, we consider 4 threads and the domain partition uses a Cartesian topology. Inside every chunk, we perform cache blocking. The z-order is applied on the threads level first, then it considers the order of the cache blocks processing and even inside a single block we can apply it. This approach guaranties an optimal reuse of data and consequently preserves data locality [Bader et al., 2006].

As mentioned before, initially the z-order technique was introduced for domains with dimensions that are power of 2. Valsalam et al., 2002 presents a more general approach that can be applied on matrices with arbitrary sizes. Their technique consists in recursively applying z-order on blocks that satisfy the power 2 condition. The results of this generalization were presented for matrix multiplications using a Strassen implementation.

3.4 ASK: Adaptive Sampling Kit

Common optimizations aim to enhance data locality through spatial and temporal cache blocking, loop tiling and padding. Most of these optimizations are machine dependent and need to be manually tuned, for instance by selecting the best blocking size and padding width. Performance modeling helps the tuning process by exploring the performance trade-offs in a large parameter space. For such exploration, we use the Adaptive Sampling Kit (ASK) [Oliveira Castro et al., 2013]. This tool uses sampling methods and gives an insight on the performance on the underlying architecture without executing all the possible combinations of parameters.

In this section, we use the isotropic FDTD implementations and explore the following parameters range:

- the grid size (X, Y, Z) where each dimension is independently selected in the

range $[768 : 1536]$ by steps of 128,

- the half stencil order p in $[1 : 8]$,
- the number of threads in $\{4, 8, 16, 32\}$,
- the number of blocks (NX, NY, NZ) respectively on X direction in $\{1, 2, 4, 8, 16\}$, Y and Z directions in $\{4, 16, 32, 64, 128\}$,
- the variant of the algorithm. The first variant, `isotropic`, corresponds to the code in figure 1. The second variant, `isotropic-split`, is the same code after loop splitting.

The possible factor combinations amount to more than 2.7 million.

3.4.1 ASK experimental setup

To produce the results presented in section 3.4.2, we build a performance model of the FDTD kernel using ASK. The target machine is the same Westmere 32-core four-socket Xeon X7550 at 2.00GHz with 128GB of RAM. Threads were distributed evenly among sockets with `KMP_AFFINITY=scatter`.

Points are sampled in 16 batches of 50 points, using three different sampling strategies: HVS, HVSrelative and Random. The surrogate model, GBM, was manually tuned on a small set of points, before starting the full ASK experiment. The selected GBM parameters are: `shrinkage`=0.05, `ntrees`=10 000, `depth`=9, `minobsinnode`=3.

In order to evaluate the error, we measure the real response of 3225 randomly selected points, which represent more than 60 hours of experiments on the 32-core machine. Since the test set is small compared to the design space, confidence intervals of the prediction error are computed using 1000 ordinary bootstrap iterations [Efron et al., 1986].

Figure 3.13 shows the RMSE and mean percentage error for the three sampling strategies. The experiment is stopped after 16 sampling steps to show ASK capability of building a performance model with a limited number of samples. HVS RMSE error curve seems to be still progressing. The most accurate model is built using HVS and GBM with a final mean error of 7.71% and an RMSE error of 4.14. The difference between HVS and random sampling in this experiment is small. Our intuition is that the design space has a high variability and gives few optimization opportunities to HVS which benefits mostly from flat regions. HVSrelative RMSE error is higher than the other methods. Since HVSrelative optimizes for relative error, this result is not surprising. However, HVSrelative does not show relative error improvement compared to the other methods in this experiment.

A GBM performance model of the FDTD kernel is built using the HVS 800 points sampling. Next section, uses this model to study the interactions between the different parameters of the FDTD kernel.

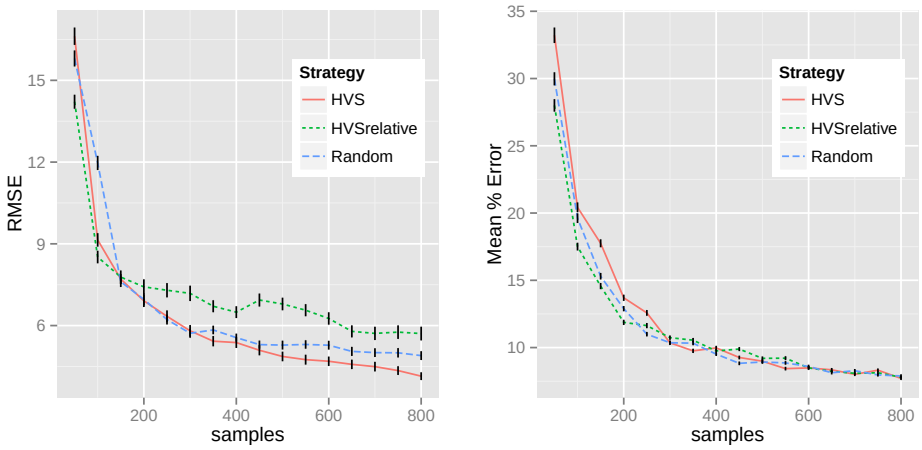


Figure 3.13 Root mean square error and Mean percentage error for the three tested strategies in the FDTD case study. The error is evaluated against a randomly selected test set of 3225 points. The vertical black lines show the bootstrap confidence intervals.

3.4.2 Performance characterization of stencil computation

The GBM model offers a useful feature that allows sorting the model factors by their relative influence. Figure 3.14 shows the relative influence of the different factors considered in our experiment. The relative influence is computed using the method proposed by Friedman in [Friedman, 2001] and determines how much a given variable affects the response in the GBM model. Dominant factors of performance are the order of the stencil, the code variant, number of blocks on X and Y and the number of threads. Figure 3.14 gives an insight on the parameters to consider in priority to enhance performance. The following paragraphs give more details on the way these parameters affect performance. The metric used is the number of cycles required per lattice update.

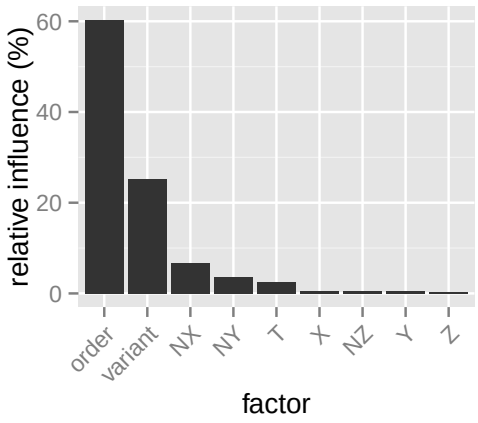


Figure 3.14 Relative influence of input factors in the FDTD kernel. The influence determines how much a factor affects the response. For the GBM model, it is computed as described in Friedman, 2001.

The Variant Influence We consider two variants of the FDTD implementation. The first variant, `isotropic` in figure 1 computes the Laplacian in a triple nested loop. In the second variant, `isotropic-split`, the inner loop is split into p smaller loops. Each split loop corresponds to one of the b_p factored blocks. Figure 3.15 shows that the number of cycles per update increases for both variants as the stencil order increases. Yet, for high stencil orders with $2.p > 10$, the `isotropic` variant is significantly more costly than the `isotropic-split` variant. The goal of loop splitting is to lower the register pressure and the number of concurrent memory streams. It is particularly effective on large loop bodies, such as the ones needed by high-order stencils.

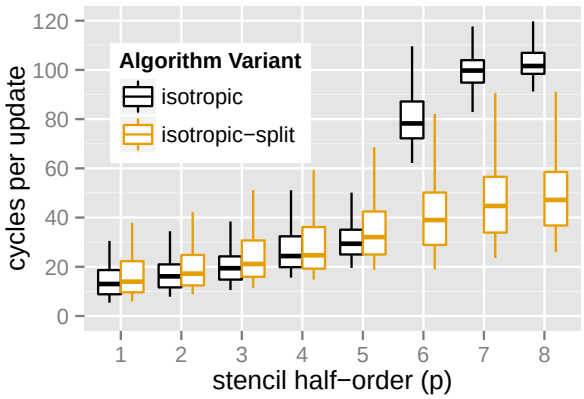


Figure 3.15 Performance of the `isotropic` and `isotropic-split` code variants. For p larger than five, the isotropic-split version is significantly faster.

The Blocking Influence This section studies the impact of the spatial cache blocking on performance in the `isotropic-split` variant with $p = 4$. Results are similar for other configurations. As seen in section 3.3.3, cache blocking aims to increase data locality by reusing data before eviction from the cache. The loop illustrated in figure 1 can be blocked across the three dimensions.

Figure 3.16 illustrates the impact of blocking on the innermost dimension X. The grid is tiled with blocks: increasing the number of blocks reduces each block size. Performance deteriorates as the number of blocks on dimension X increases. Indeed, for small block sizes on X, the hardware prefetcher streams additional data, which are evicted from the cache before being used. Therefore, using small X block sizes result in an increase of memory traffic. The number of blocks in the X dimension should be kept low.

On the other hand, the outer Y and Z loops should be blocked to make the data working set of all the threads fit the cache. Figure 3.17 shows the correlation between the number of Y blocks and the number of threads. For high number of threads, configurations with a high number of Y blocks are the best. All the threads in the same socket share the same Last Level Cache (LLC). As the number of threads increases, the cache budget per thread decreases requiring smaller block sizes. Blocking across Z also helps, but the pay-off is smaller since it exposes less data reuse. Similar conclusions can be found in other studies on performance optimization of stencil computations such as [Nguyen et al., 2010; Rivera et al., 2000].

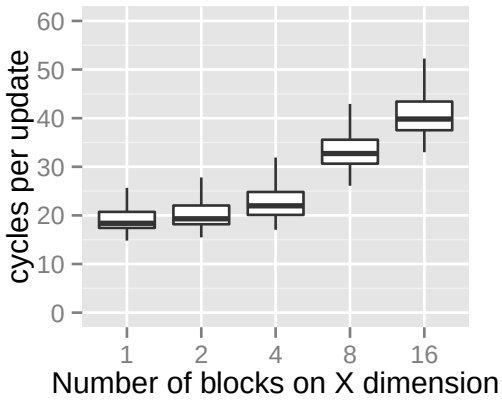


Figure 3.16 Performance of different X blocking configurations. The size of the blocks is inversely proportional to the number of blocks. Large blocks sizes across X exhibit the better performance.

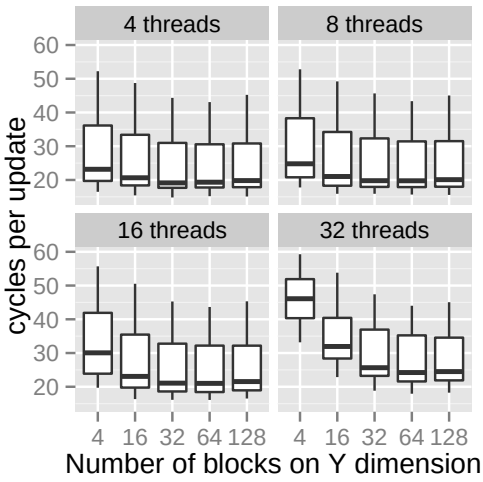


Figure 3.17 Performance of different Y blocking configurations. For a high number of threads, reducing the Y block size improves performance.

Scalability This section studies the strong scalability of a stencil of order 8 ($p = 4$) with the best variant and blocking parameters determined in the previous analysis. The selected parameters are `isotropic-split`, $p = 4$, $NX = 1$, $NY = 128$, and $NZ = 32$.

Figure 3.18 shows the scalability for two different grid sizes.

The kernel scales well up to 16 threads. ASK pinpoints a potential scalability problem, at 32 threads the speedup is around 26. Since the results are extrapolated from the model, we cross validate this result by direct measurement. Despite some small local discrepancies, the model predicts the general trend.

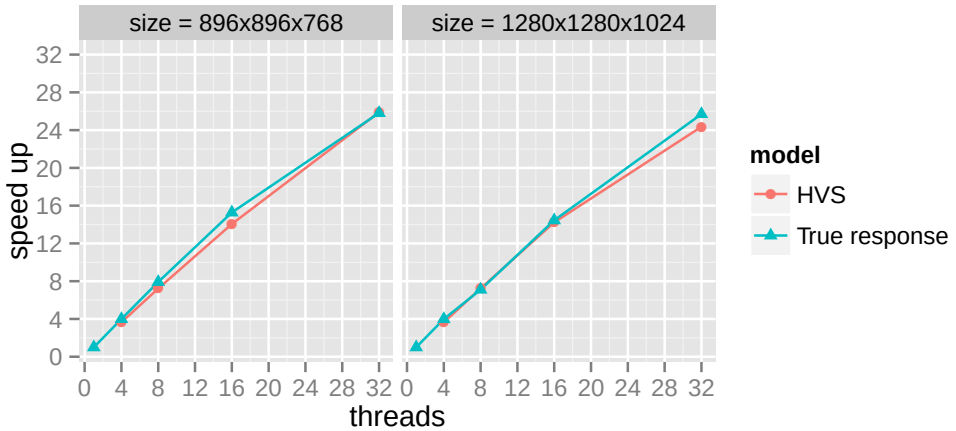


Figure 3.18 Scalability for the `isotropic-split` implementation with half order $p = 4$, $NX = 1$, $NY = 128$, and $NZ = 32$. The speedup is computed using the single thread performance with the same parameters.

The model generated by ASK allowed us to tune the variant, the blocking factor, and to detect a scalability problem. The analysis presented here reaches similar conclusions to previous studies on stencil performance. Our results with ASK show that adaptive sampling is a valid approach for real application performance exploration.

ASK permitted to identify the most influencing parameters This chapter focuses on optimizations applied to FDTD applications affecting data accesses pattern and data reuse. Based on the algorithm describing lattice updates, we derived a performance model that accounts for theoretical peak performance of the machine. This gives us bounds on the expected performance.

ASK permitted us to identify the most influencing parameters for FDTD. The order of the stencil studied has the greatest impact on performance followed by the loop split and the size of the cache block on the X dimension.

In the next chapter, we define more precisely the interaction between architecture and the FDTD applications.

CHAPTER 4

Memory Bandwidth Cost Model for FDTD

In the previous chapter, we gave an insight on the optimizations suitable for stencil-based applications class. In this chapter, we explore these optimizations on actual codes in order to highlight the architectural features that impact the performance and to quantify their influence.

We consider for our experiments isotropic implementations and we are interested in their multi-threaded versions. Our aim is the use of the full capacity of a computing node in order to delimit the scope of the applied optimizations on a node level.

For these experiments, we vary the arithmetic intensity of the applications studied while maintaining the same memory size allocated for wavefields. Hence, for the same problem size, we have more or less floating point operations.

We also bring to a focus the impact of the widening gap between floating point rates and memory. We use the reuse-distance, i.e. the distance separating consecutive references to the same element and accounting for distinct accesses, as a metric for data locality and for efficient use of the caches. This metric results in a model for performance prediction.

Experiments are performed on the Westmere machine described in table 3.2 and we use the Roofline model as described in 3.1.3 to plot the performance variations on this particular machine depending on the optimizations performed and the half stencil order p. For the peak rates, we use the peak floating point performance and the sustained bandwidth for a single socket. These values are also given in table 3.2.

4.1 Isotropic Kernel

On figure 4.1, we plot the performance in terms of Gflop/s for the isotropic implementation when using different stencil orders. The multi-threaded version runs on a single socket on the node 3.2 and uses the 6 cores available on it. Performance measurements are done using LIKWID , a lightweight tool that permits to measure hardware counters simply using model specific registers (MSRs) from user space

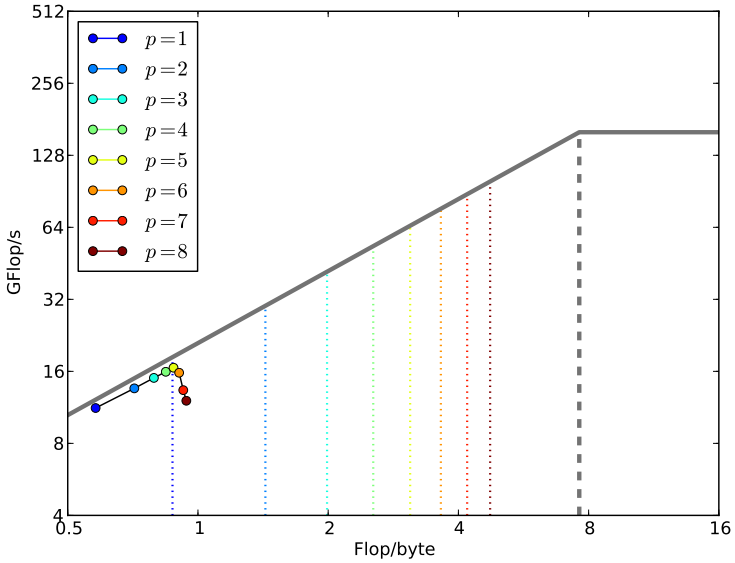


Figure 4.1 Roofline for the non-blocked isotropic implementation.

[Treibig et al., 2010]. We retrieve the GFlop/s data and the corresponding number of Bytes and floating point instructions.

We also show the theoretical arithmetic intensity through the dashed vertical lines. It is computed using values reported in table 3.1 and only accounts for the compulsory traffic. The intersections of these dashed lines with the peak performance rates of the machine represent upper bounds for the performance of the application depending on the value of the half stencil length p. Hence, the aim is to approach these values. Plotting performance measurements on a Roofline model is an easy way to figure out the possible optimizations that we can apply. It also shows the gap between the peak performance rates and the achievable performance and thus the efficiency of an implementation.

On figure 4.1, we notice that the real data measurements are rather in the *bandwidth bound* area of the Roofline model and that for almost half stencil lengths the performance on single socket is limited by the bandwidth. We also notice that for values of p greater than 5 performance decreases and it is not only limited by the bandwidth. This is due to register spilling in the case of high stencil orders. It results in increased pressure on the memory bus and increased instruction count. We opt for the split of the inner loop in algorithm 1 into smaller ones since it represents the hotspot of the application. This reduces the pressure on registers and the number of the instruction count within the loops. As a result, performance increases and is only bounded by the bandwidth as shown on figure 4.2.

We are interested in data movement optimizations and their impact on these versions of the FDTD application. We focus on the cache blocking technique and

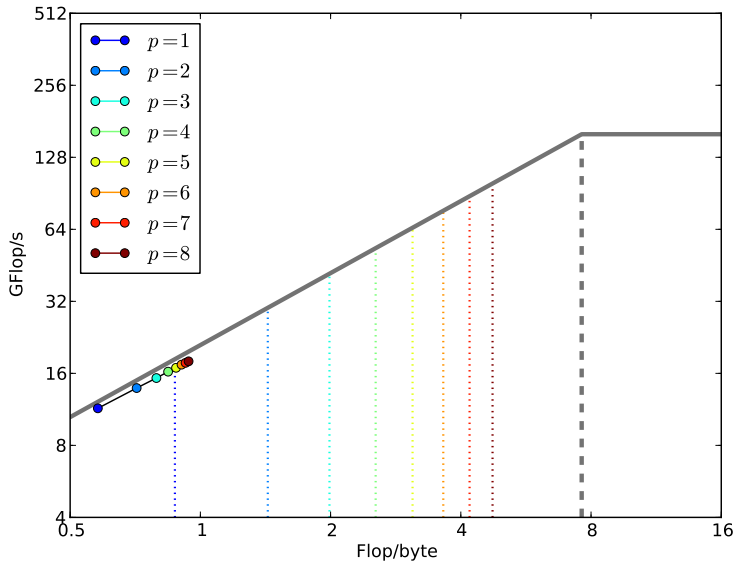


Figure 4.2 Roofline for the non-blocked isotropic implementation after loop splitting.

we choose an optimal block size. The figure 4.3 shows the impact of the blocking on the initial version of the application. We still have the drop of the performance due to register spilling for p greater than 5. We notice that the measured performance is getting closer to peak performance of the algorithm. On figure 4.4, we also have the same tendency. Efficient use of cache guaranties data locality and since the application we are studying is bounded by the bandwidth, reducing DRAM traffic results in better performance.

On figure 4.4, we notice that even after applying the cache blocking in order to enhance data locality, we still have extra DRAM traffic. This is illustrated by the discrepancy between the theoretical value presented by the dashed line and the real data collected. Our goal is to explain this gap between the two values. We use the reuse distance histogram in order to compute this extra DRAM traffic for both blocked and non-blocked versions of the application.

4.2 Performance Model for Extra DRAM Traffic

The previous section showed that the FDTD application is bounded by bandwidth and that optimizations affecting data movements are extremely important in order to increase the overall performance on a node level. As shown on figure 4.4 cache blocking has a great impact on data locality in the case of FDTD applications.

In the following paragraphs, we aim to predict the amount of the extra DRAM traffic generated. We use the data reuse histogram in order to quantify capacity misses rate.

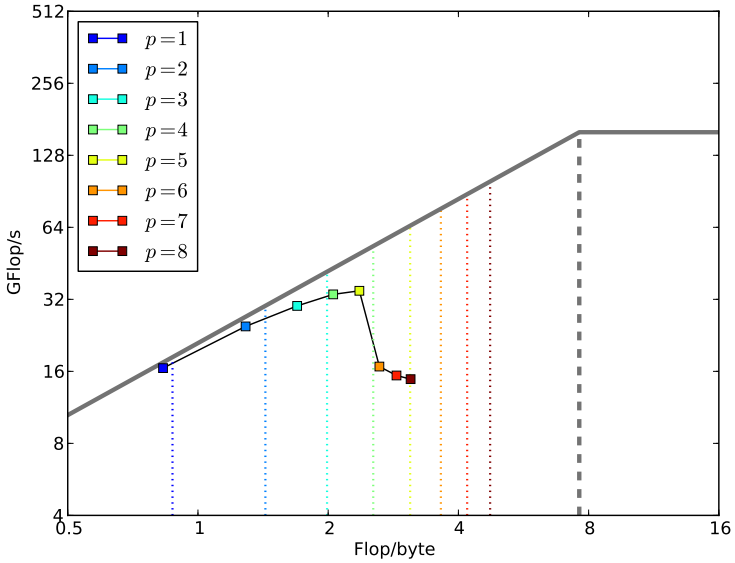


Figure 4.3 Roofline for the blocked isotropic implementation.

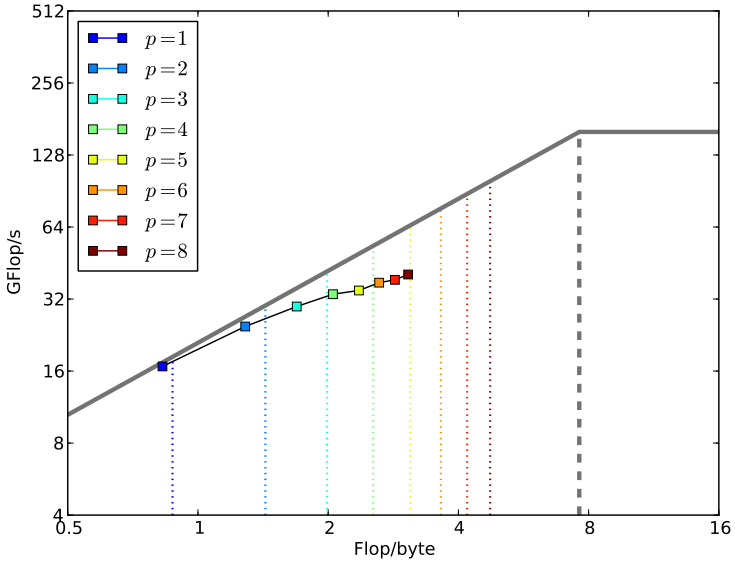


Figure 4.4 Roofline for the blocked isotropic implementation after loop splitting.

4.2.1 Data Reuse Histogram

Reuse distance is defined by Beyls et al., 2001 as the number of unique memory locations separating the last use and the reuse of an element. This metric is also known as the LRU (Least Recently Used) stack distance in reference to stack algorithms for cache management studied by Mattson et al., 1970. On this basis and

considering a fully associative cache, memory trace is represented as a stack where the last data requested is placed on the top of it. Hence, the reuse distance is the distance separating consecutive references to the same element and accounting for distinct accesses. Table 4.1 is an illustration of how this metric is computed. The reuse distance of an element that has never been used before is equal to ∞.

Time:	1	2	3	4	5	6	7	8	9	10
Access:	d	f	a	b	a	d	e	f	a	b
Reuse distance:	∞	∞	∞	∞	1	3	∞	4	3	4

Table 4.1 An example of reuse distance computation considering memory address granularity.

Reuse distance is a metric that enables performance prediction independently of the hardware used. It rather describes data accesses pattern in the algorithm implemented. Many studies have used this metric to evaluate and analyze data locality [Shen et al., 2003; Ding et al., 2003; Marin et al., 2004; Zhong et al., 2007].

Reuse distance gives another definition of the cache misses introduced by the 3C's model presented in paragraph 3.3.3. In the case of a fully associative LRU cache:

- *Compulsory miss* corresponds to a reuse distance equal to ∞.
- *Capacity miss* happens when the reuse distance is greater than the cache size.
- *Conflict miss* occurs for a reuse distance smaller than the cache size.

4.2.2 Applying the Reuse Distance Histogram to FDTD

Since capacity misses dominate the cache misses rate and since advanced compiler optimizations reduce efficiently conflict misses, we use reuse distance metric to compute capacity miss rates for the FDTD applications.

According to the algorithm 2, updating a value of the wavefield u requires to load its previous value and those of arrays containing velocity and Laplacian and to store the latest value of u.

Equation 4.1 defines the *Extra DRAM traffic* as the ratio between the actual DRAM traffic and its theoretical value as counted in table 3.1.

$$\textbf{Extra DRAM traffic} = \frac{load\ (u) + load\ (others) + store\ (u)}{3+1} \tag{4.1}$$

We consider the figure 4.5 describing the accesses by a single thread when it updates a single value within the block assigned to it. Therefore, we can estimate the DRAM traffic per a lattice update (LUP).

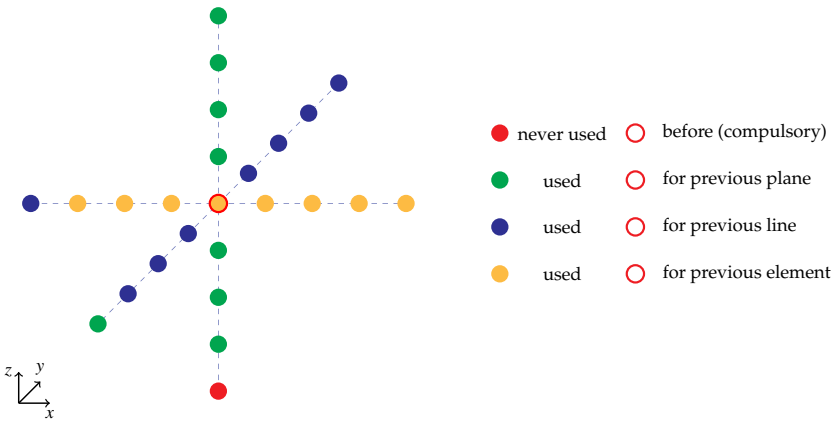


Figure 4.5 Data access patterns for stencil based applications. We consider XY planes and lines on the X direction.

The figures 4.6 and 4.7 plot the capacity misses per a lattice update as a function of the reuse distance $\mathcal{D}$. We have two figures because we consider both blocked and non blocked cases.

If we use cache blocking, we have a single compulsory miss. On the other hand, if we are out of the cache, we expect $2p+1$, where p is the half stencil length.

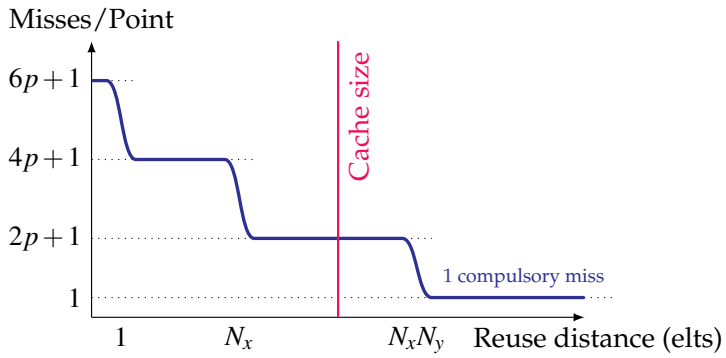


Figure 4.6 Data reuse histograms without cache blocking. N_x and N_y refer to the grid size on the X and Y directions.

Based on the algorithm 2 and according to the implementation of the FDTD application we are studying, we need to store the latest value of u, load 2 elements from other arrays (velocity and Laplacian) and load values of u at time steps t and $t-1$. The system 4.2 summarizes the number of elements needed for a lattice update. The variable $\mathcal{D}$ represents the reuse distance.

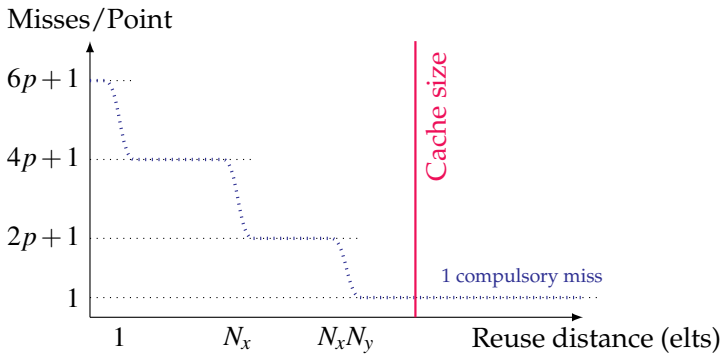


Figure 4.7 Data reuse histograms when cache blocking is used. In this case, N_x and N_y represent the size of cache blocks on the X and Y directions.

$$
\begin{aligned}
store\ (u) &= 1 \\
load\ (others) &= 2 \\
load\ (u) &= 1 + f(\mathcal{D})\ \ where\ \ f(\mathcal{D}) = \begin{cases} 0 & \text{if } \mathcal{D} > N_x N_y \\ 2.p & \text{if } \mathcal{D} > N_x \\ 4.p & \text{if } \mathcal{D} > 1 \\ 6.p & \text{else.} \end{cases}
\end{aligned}
\tag{4.2}
$$

Replacing the values of system 4.2 in the expression 4.1 results in an estimation of the *extra DRAM traffic* equal to the expression 4.3 where $\mathcal{D}$ is the reuse distance.

$$\textbf{Extra DRAM traffic} = \frac{4 + f(\mathcal{D})}{4} \tag{4.3}$$

For an accurate prediction, we consider the sizes of the arrays involved in the compute of a lattice update. Laplacian and velocity arrays has the same size of the wavefield $nfield = N_x\ N_y\ N_z$ while we need extra elements on boundaries for the wavefield u.

The compute grid is equal to $ngrid = (N_x + 2\ p)\ (N_y + 2\ p)\ (N_z + 2\ p)$. For each time step we need to load $nfield$ elements for velocity and for Laplacian and $ngrid$ elements for u. We also need to store $nfield$ elements of the wavefield u. As a result, the number of bytes required to update a single lattice at each time step are computed in equation 4.4.

$$bytes\_per\_lup = sizeofreal\left(\frac{nfield + 2\ ngrid + ngrid}{ngrid} + f(\mathcal{D})\right) \tag{4.4}$$

We assume that the cache size can hold N_x elements at least but is not large enough to hold $N_x\ N_y$ elements. We are in the case illustrated in figure 4.6 which means that $f(\mathcal{D}) = 2.p$. Hence, the number of bytes per a lattice update is equal to the expression 4.5.

$$bytes\_per\_lup = sizeofreal\left(3 + 2\ p + \frac{nfield}{ngrid}\right) \tag{4.5}$$

On the figure 4.8, we compare the measured DRAM traffic per a lattice update with the predicted values using the model as described in equation 4.5. This comparison concerns both the blocked and the non-blocked cases. We first notice the discard between these two versions in terms of the generated DRAM traffic which confirms the positive effect of the cache blocking on it. We also notice that the theoretical model in equation 4.5 predict almost perfectly the DRAM traffic in the non-blocked case. On the contrary, we overestimate the DRAM traffic using this model when we have cache blocking.

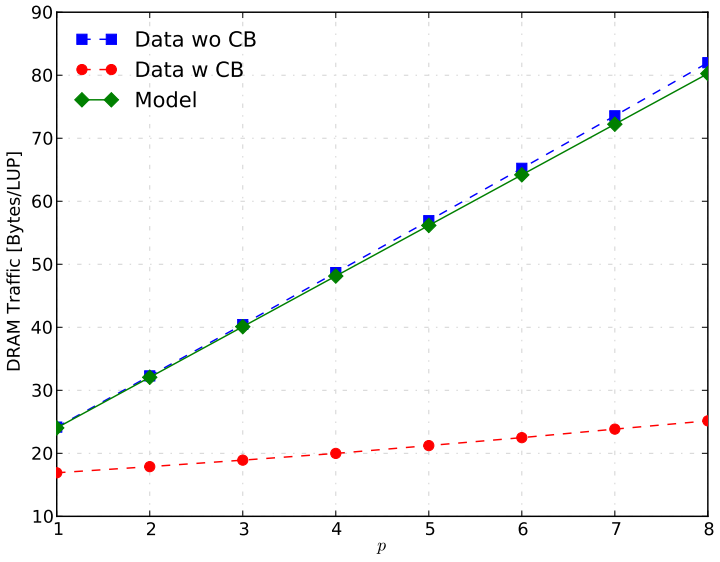


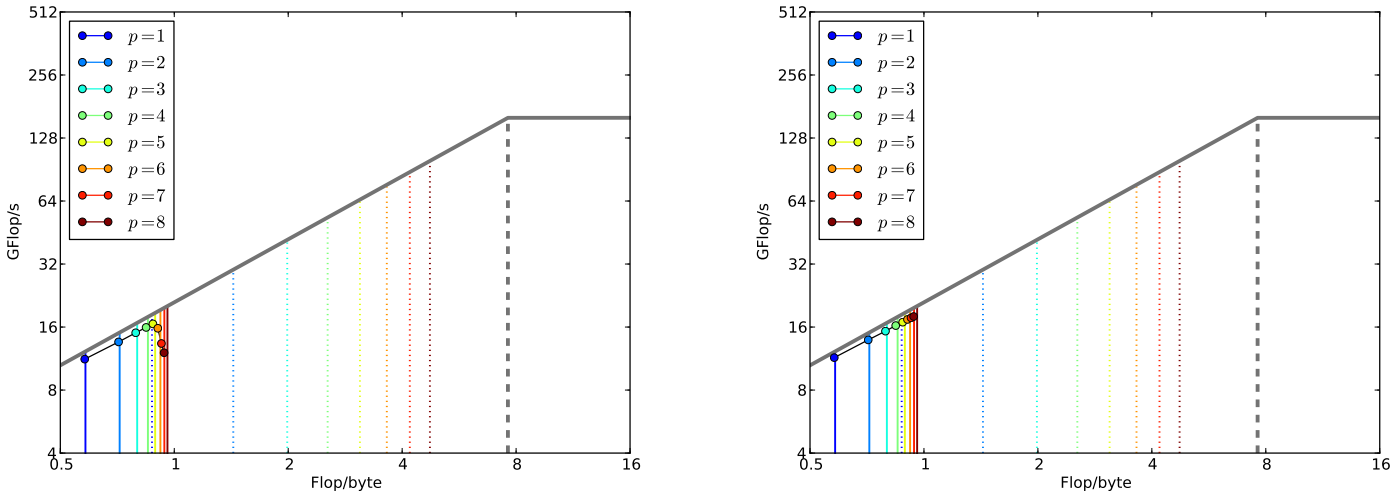
Figure 4.8 Extra DRAM traffic prediction for different stencil orders

Let us place the theoretical model in equation 4.5 on the previous Rooflines. This gives us the figures 4.9 and 4.10. We keep the dashed lines that compute the arithmetic intensity of the application for compulsory traffic and we introduce the thick lines for the model 4.5. As in figure 4.8, we have an overestimation of the DRAM traffic when cache blocking is used and we have a better estimation for non-blocked FDTD application.

We need to adjust this discard in the case we use cache blocking since it gives better performance. The following paragraph is dedicated to the improvement of the model.

4.2.3 Updated Formulation of the Performance Model

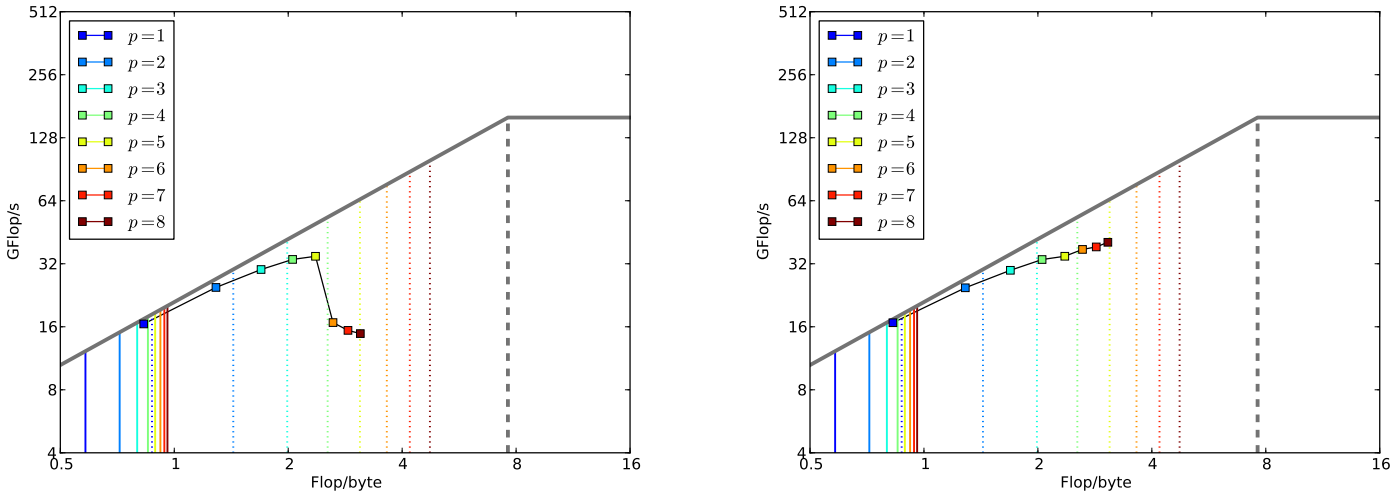
In this paragraph, we introduce the needed adjustments to adapt the model in equation 4.5 to the blocked case. We need to take into account the data reuse within the block. At each time step and for single lattice update in the block, we load $nblock = B_x\, B_y\, B_z$ elements of the velocity and Laplacian arrays and $ngridblock = (B_x + 2\,p)\,(B_y + 2\,p)\,(B_z + 2\,p)$ elements of u. We store $nblock$ elements of u. This



(a) No loop split

(b) With loop split

Figure 4.9 Good estimation of the DRAM traffic without cache blocking.



(a) No loop split

(b) With loop split

Figure 4.10 Overestimation of the DRAM traffic with cache blocking.

results in equation 4.6.

$$bytes\_per\_lup = sizeofreal\left(3 + 2\,p + \frac{nblock}{ngridblock}\right) \tag{4.6}$$

The DRAM traffic on figure 4.11 shows a better estimation than in figure 4.8 for

cache blocked version with the updated model in equation 4.6.

As in the previous paragraph, we compare the updated model 4.6 to the actual data on Rooflines. On the figures 4.12, we also notice that in the blocked case, we decrease greatly the overestimation of the DRAM traffic for both split and not split versions of the FDTD application.

For high stencil orders, we still have a slight overestimation of the DRAM traffic that we can see on figure 4.11 and on Rooflines 4.12.

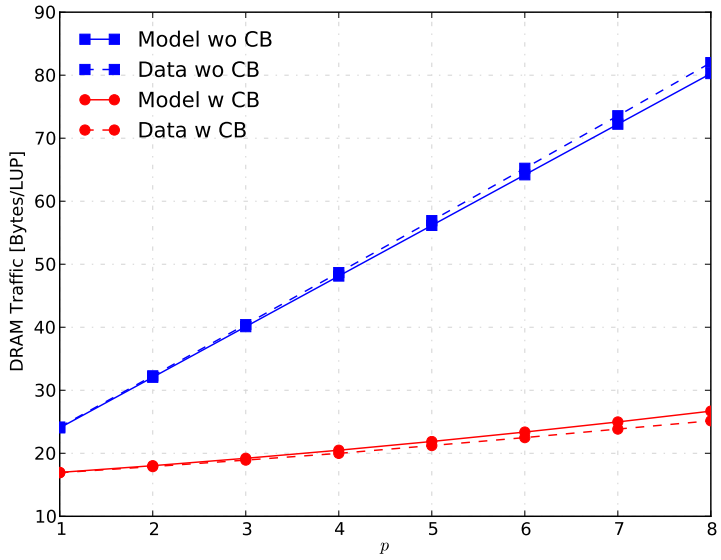
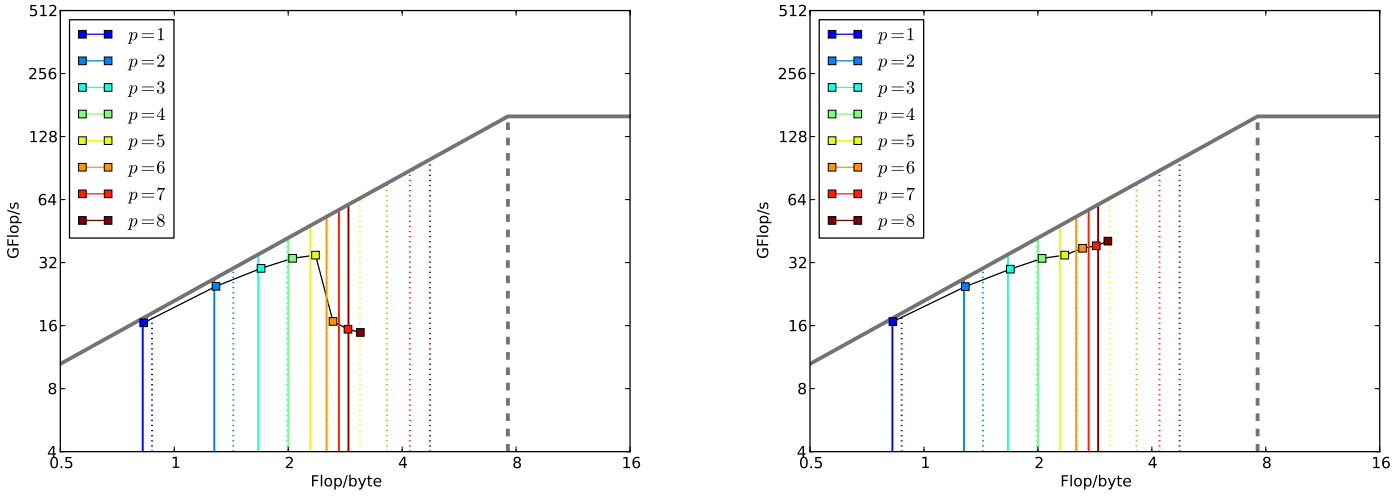


Figure 4.11 Extra DRAM traffic prediction for different stencil orders after updating the model.

In this chapter, we applied the Roofline model in order to characterize the FDTD applications. We identified the memory bandwidth as a limiting factor for this application class even for high order schemes. Applying optimizations affecting the data access pattern and data reuse showed a great enhancement of the overall performance.

The reuse distance histogram applied to FDTD applications resulted in a model predicting their DRAM traffic. We had to adapt this model to kernels using cache blocking techniques. This model was validated using all stencil order in isotropic media.



(a) No loop split

(b) With loop split

Figure 4.12 Better estimation with cache blocking after updating the model.

CHAPTER 5

FDTD Applications on Manycore Architectures

This chapter gives a study of the finite-difference time-domain (FDTD) applications on manycore architecture. We use an Intel Xeon Phi coprocessor since it has two main characteristics of a node in an Exascale system as described in chapter **??**. We confront our code to an increasing amount of concurrency since a Knights Corner (KNC), Intel's first commercial product hosting a Xeon Phi coprocessor, has up to 61 cores. We also identify the possible difficulties related to heterogeneity since we can use both the host and the CPU.

In this chapter, we first start with an introductory section describing this new architecture and the new hardware features that characterize it. The following sections describe the porting of the FDTD applications on KNC. We conduct a single node study and focus on the challenges we faced in order to benefit from this highly parallel architecture. Communications aspects and load balancing tasks were highlighted in the last section of this chapter using a KNC cluster. The work presented in this chapter is done in preparation for the porting of the Reverse Time Migration (RTM) application on KNC. This upstream study is important since the computational core of the RTM we are considering uses FDTD in isotropic and anisotropic media.

5.1 Intel Many Integrated Core Architecture

We use a Knights Corner (KNC) prototype of Intel Xeon Phi 7120. This prototype hosts 2 Intel Many Integrated Core (MIC) coprocessors connected to a 2-socket Xeon E5-2670 codename Sandy Bridge EP host via PCIe. Each MIC coprocessor has 61 cores with 4 threads each for a total of 244 threads. Two instructions of the same hardware context can not execute on the core. Using a single thread per core means that 50% of the compute capacity of the coprocessor is not used. The width of the SIMD vector is equal to 512-bit against 256-bit on Sandy Bridge which makes this architecture more adequate to highly vectorized and parallel codes. On this coprocessor, each core has only 2 cache levels L1 and L2. The last level caches are connected via a double channel ring interconnect. There are 8

memory controllers each supporting 2 GDDR5 channels. These hardware features are summarized in table 5.1 where we give those of Sandy Bridge also. Figure 5.1 is a high level illustration of the Intel MIC architecture.

	Knights Corner	**Sandy Bridge**
Number of cores	61	16
Threads per core	4	1
Cache L1/L2/L3	32KB/256KB/ -	32KB/256KB/20MB
Memory	16 GB	60 GB
CPU frequency	1.238 GHz	2.6 GHz
SIMD width	512-bit	256-bit
Peak performance SP	2416.5 Gflop/s	665.6 Gflop/s
Energy TDP	300W	115W

Table 5.1 Hardware features of an Intel Knights Corner prototype.

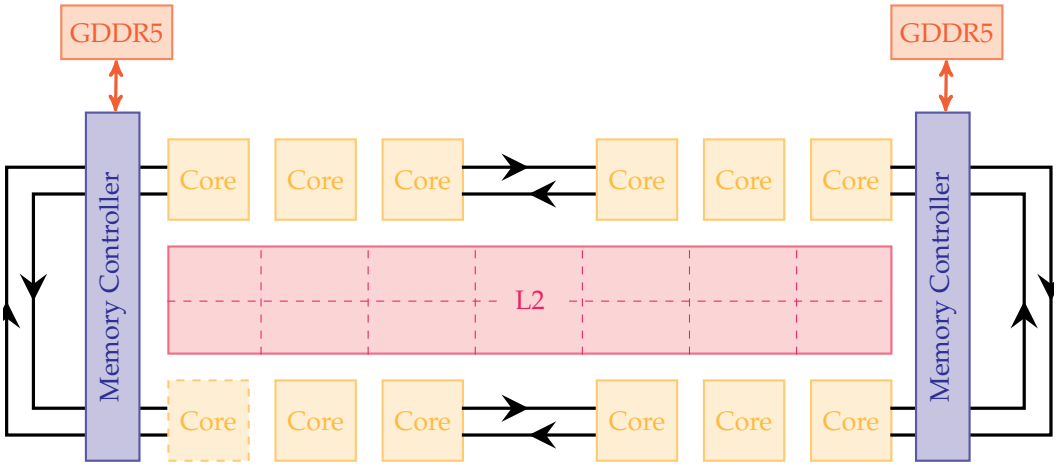


Figure 5.1 Block diagram of Intel MIC coprocessor.

Component	**Version**
OS on host	Red Hat Enterprise Linux Server 6.1
Kernel	2.6.32-220.el6.x86_64
Manycore Platform Software Stack (MPSS)	3.1.2
Compiler	Intel 14.0.1
MPI	Intel 4.1 update 3

Table 5.2 Software stack on the Knights Corner prototype.

Characteristic of the software stack on the prototype we used are detailed in table 5.2. The nodes topology is illustrated in figure 5.2. We used the Portable

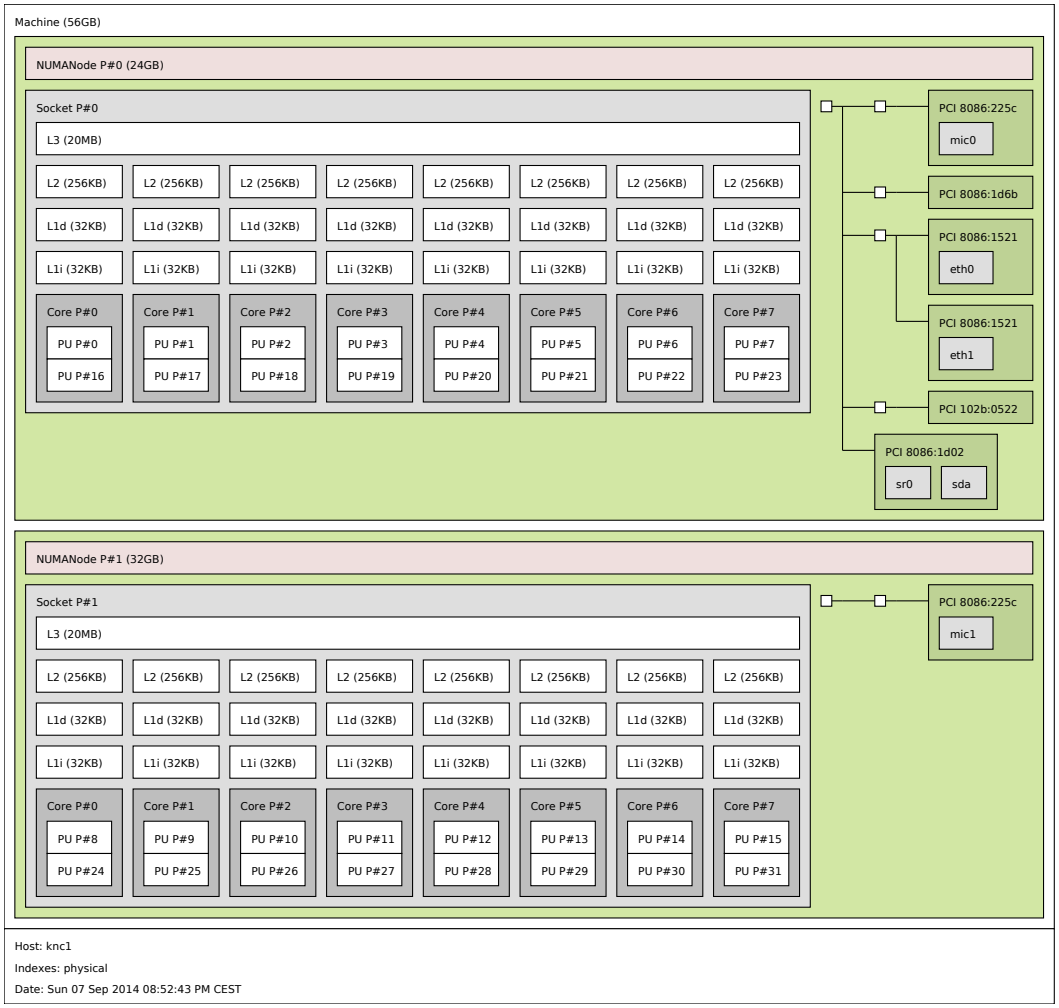
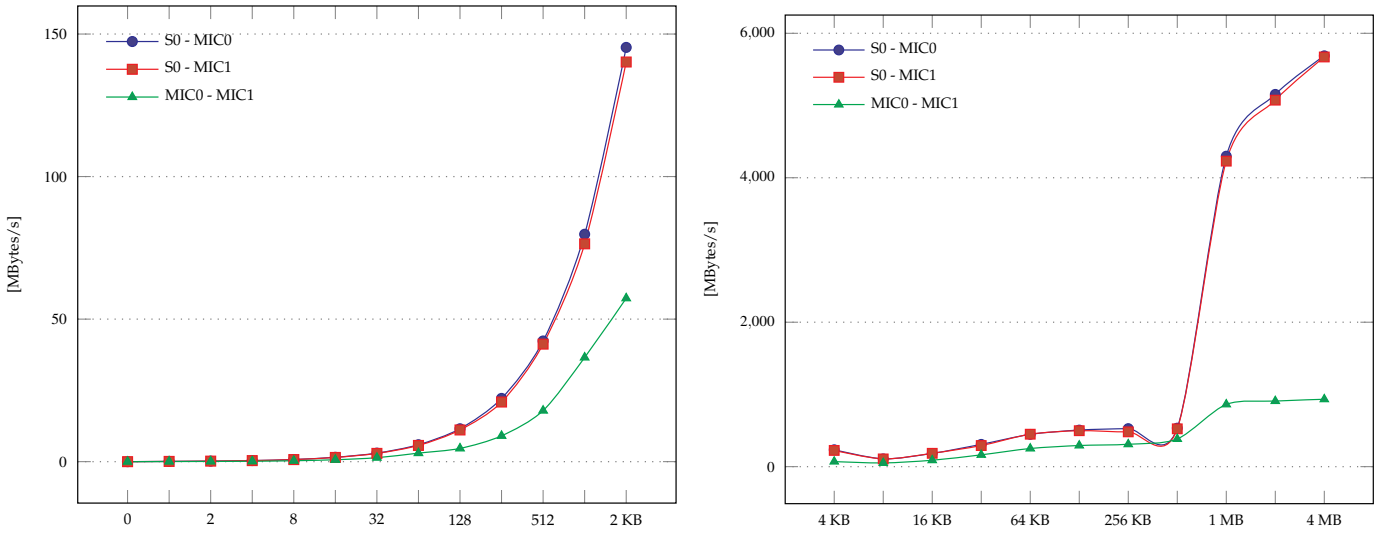


Figure 5.2 The phi, denoted *mic0* and *mic1* are respectively connected to NUMA domain 0 and 1 corresponding to the two sockets.

Hardware Locality (hwloc) tool [Broquedis et al., 2010] to generate this figure. As we can see, the coprocessor `mic0` is connected to the socket 0 of the host. This socket 0 is also connected to the Gigabit Ethernet interfaces `eth0` and `eth1`. The coprocessor `mic1` is connected to socket 1.

We measured the unidirectional bandwidth using the pingpong provided in the Intel MPI Benchmark (IMB) suite. Figure 5.3 reports these values. We notice that the bandwidth between socket 0 and the coprocessor `mic0` and `mic1` is the same. We had the same values for socket 1. On the KNC prototype 5.1, we use the x16 Gen2 PCIe interface which has a maximum theoretical bandwidth of 8 GBytes/s. The bandwidth measured between the host and the coprocessor for 4 MBytes messages is equal to 5.6 GBytes/s.

In the case of communications between the coprocessors `mic0` and `mic1`, the bandwidth is only 933 MBytes/s for 4 MBytes messages.



(a) Small messages

(b) Large messages

Figure 5.3 Bandwidth measurements for intra-node communications.

As explained in chapter 1, DOE reports advise the use of concurrency as a response to power efficiency of Exascale machines. Data movements between nodes contribute greatly to power consumption and increasing the number of cores within the node reduces these movements. As a result, we increase the on-node parallelism. DOE reports also predict that at least the first versions of an Exascale machine will probably use heterogeneous architectures. Concurrency and heterogeneity are found in Intel Knights Corner. We explore the impact of these characteristics on the FDTD applications and on the RTM application in chapter 6.

5.1.1 Performance Gain Expectations

In this paragraph, we evaluate the theoretical gain when using the MIC co-processor compared to Sandy Bridge. The table 5.3 gives the theoretical DRAM bandwidth and the sustainable bandwidth for both architectures. The achievable bandwidth for MIC was measured using the STREAM Triad benchmark based on Intel recommendations for the compiler knobs [Raman, 2013].

- `-opt-prefetch-distance=64,8` : 64 cache lines prefetched for L2 cache and 8 cache lines for L1 cache.
- `-opt-streaming-cache-evict=0` : all cache line evicts turned off.
- `-opt-streaming-stores always` : streaming stores enabled.
- `-DSTREAM_ARRAY_SIZE=64000000` : array size at least equal to 4x the size of the sum of all L2 caches.

The results of this benchmark are illustrated on figure 5.4. These results are given when Error-Correcting Code (ECC) memory is activated or not. We notice

that ECC memory results in an overhead and impacts the overall memory bandwidth. We keep the ECC memory activated for our stencil-based computations which avoids us bit-flip and thus numerical issues.

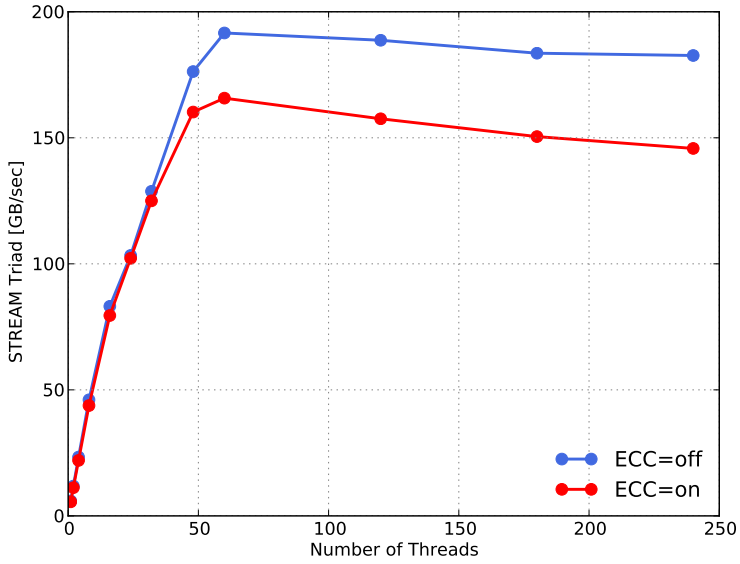


Figure 5.4 Results of STREAM Triad benchmark on Intel MIC coprocessor. The best results are obtained when the ECC memory is disabled.

	Channels	Frequency[MHz]	β_t[GB/s]	β_s[GB/s]
Sandy Bridge	4	1600	102.4	78
Knights Corner	16	2750	352	195/165

Table 5.3 Theoretical and sustainable bandwidth measurements for MIC and Sandy Bridge Architectures.

For compute bound applications, we consider the single precision peak performances as reported in table 5.1. We also assume that a compute bound application on Sandy Bridge remains compute bound on Knights Corner. This ratio is equal to

$$\frac{P_{mic}}{P_{snb}} = \frac{2147\ Gflop/s}{665\ Gflop/s} = 3.2$$

We make the same assumption in the case of bandwidth limited applications and consider the expression 3.4 defined in chapter 3 for stencil computations. The ratio of performances is equal to the ratio of bandwidth values. For Knights Corner, we activate the ECC memory. Based on the values of table 5.3, we have this ratio

$$\frac{P_{mic}}{P_{snb}} = \frac{\beta_{mic}}{\beta_{snb}} = \frac{165\ GB/s}{78\ GB/s} = 2.1$$

Based on these ratios, we expect for stencil-based computations a speedup comprised between 2.1x and 3.2x relatively to 2-socket Sandy Bridge. In the following paragraph, we port our isotropic and anisotropic applications on Knights Corner and explore the real performance gain on this new architecture.

5.1.2 Programming Models on MIC

Intel MIC offers the possibility to use three different programming methods.

- Native mode : the whole application runs directly on the coprocessor. The host is not involved in this case.
- Offload mode : the application runs on the host and some parts are executed on the coprocessor. Data are transfered via the PCIe.
- Symmetric mode : both the coprocessor and the host are used simultaneously as two independent nodes.

5.2 Single-node Implementation of FDTD Applications

Throughout this section, we study the impact of the manycore architecture on the FDTD applications. In this chapter, we only investigate the impact on performance at the node level. We also consider the impact of the programming models provided by Intel Knights Corner and presented in paragraph 5.1.2.

5.2.1 FDTD Implementations Without Absorbing Conditions

For our FDTD implementations, we mainly focus on native and symmetric modes. The offload implementation induces an overhead due to transfers of the whole wavefield and the allocation of memory on the coprocessor side at each time step.

5.2.1.1 Native Implementation

For the native implementation on MIC, we use a multi-threaded version of the FDTD applications. In this study, we focus on two aspects of these implementations. We are first interested in the performance behavior of the stencils of order 8 in space and 2 in time since they are widely used in oil and gas production code. We also consider the impact of the variation of the stencil order on performance when using MIC. We describe the optimizations we need to perform on MIC in order to benefit from its new hardware features.

Figures 5.5 and 5.6 depicts the impact of the multi-threading on the performance of the isotropic and TTI implementations of the FDTD applications. This experience highlights the positive impact of hyper-threading on the FDTD implementations when we have 2 threads per core compared to the performance when

a single thread is used per core. This is due to the hardware context switch in the case of 2 threads. We have almost 50% gain in performance for both isotropic and anisotropic cases. For example in the isotropic implementation, performance increases from 122 Gflop/s for 1 thread to 180 Gflop/s for 2 threads.

When we add more than 2 threads per core, performance decreases. We have a slowdown of 17% when we use 4 threads per core for both isotropic and TTI implementations. This decrease of the performance is due to contention in the L2 cache on MIC since it is shared by the 4 threads.

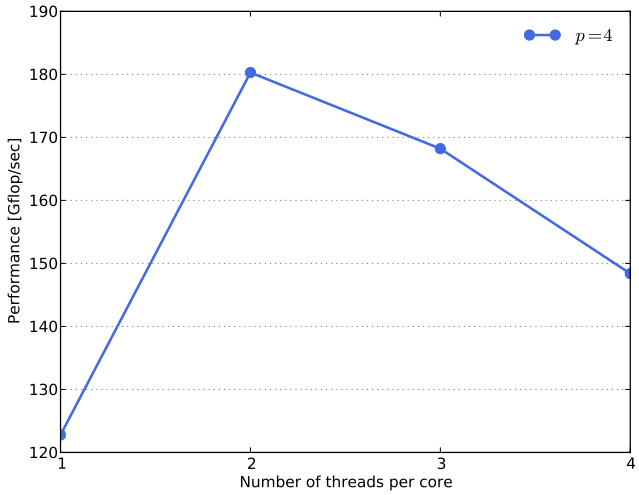


Figure 5.5 Performance of the isotropic kernel using an order 8 in space in native mode as a function of the number of threads per core on Intel MIC architecture.

Figures 5.7 and 5.8 respectively illustrate the performance variation of the isotropic and TTI implementations when the half stencil orders in space vary from 1 to 8 and when we vary the number of threads per core also. Increasing the order of the stencil in space increases the arithmetic intensity of the application while maintaining the same memory requirements for the same grid size. We notice that for both implementations, performance drops for higher stencil orders ($p \in 5..8$).

Vectorization. In this paragraph, we show the impact of vectorization on the FDTD kernels using the Intel MIC architecture. On figures 5.9 and 5.10, we illustrate the performance when we deactivate vectorization using the options `-no-vec` and `-no-simd` during compilation. We compare this performance with the result we obtain when we place the `SIMD` pragma above the inner loops. We also give the results for different thread numbers per core. Vectorization can increase performance up to 17x in the isotropic case on Intel MIC. On Sandy Bridge, we have a speedup of 3.8 as shown on figure 5.11. For TTI, we have an increase of 8x on MIC and an increase of 3x on Sandy Bridge. Due to wider SIMD vectors on MIC, vectorization impact is more noticeable on this architecture that on Sandy Bridge

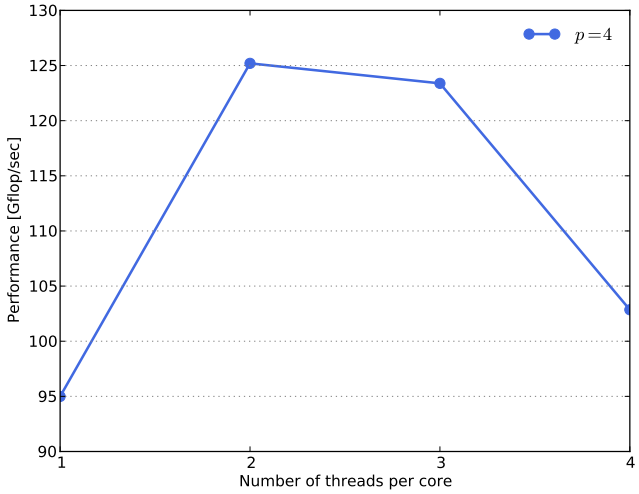


Figure 5.6 Performance of the TTI kernel using an order 8 in space in native mode depending on the number of threads per core on Intel MIC architecture.

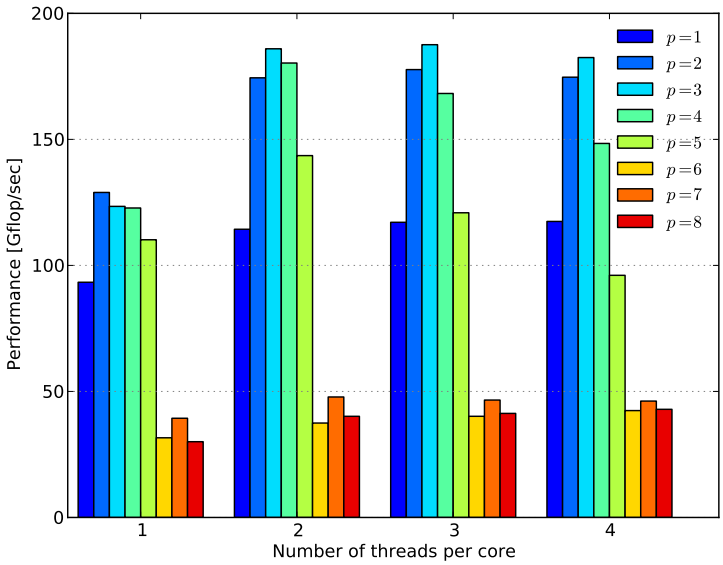


Figure 5.7 As we increase the order of the stencil, we increase the number of arithmetic operations to compute the Laplacian of the wavefield. We notice that hyper-threading has a positive impact on the isotropic implementation.

for example. This confirms the need to have a highly vectorized code in order to benefit of the MIC.

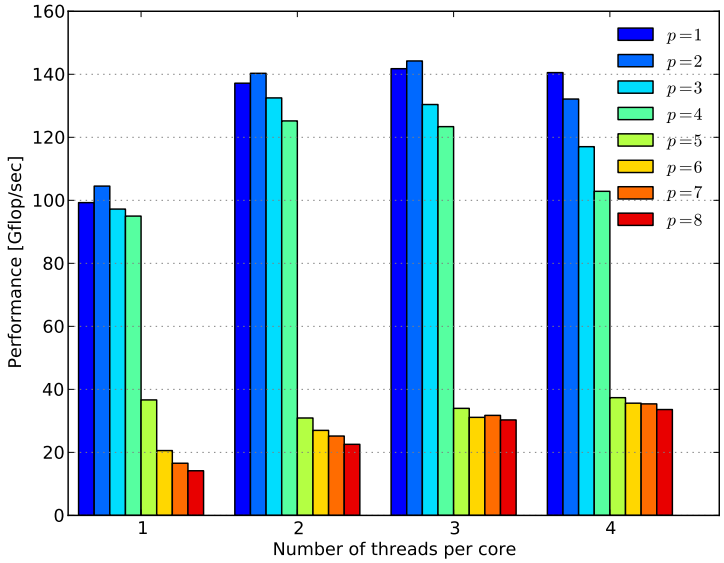


Figure 5.8 TTI Implementation using different orders in space while varying the number of threads per core in native mode on Intel MIC.

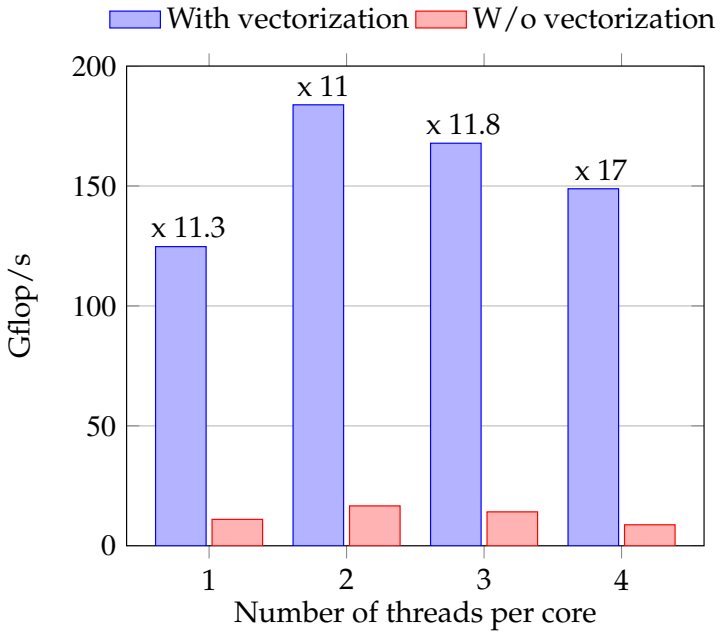


Figure 5.9 Vectorization impact on the isotropic implementation on Intel Knights Corner.

5.2.1.2 Offload Implementation

In offload implementation, the computational part of the application is computed on the coprocessor. This implies multiple transfers of the data during the

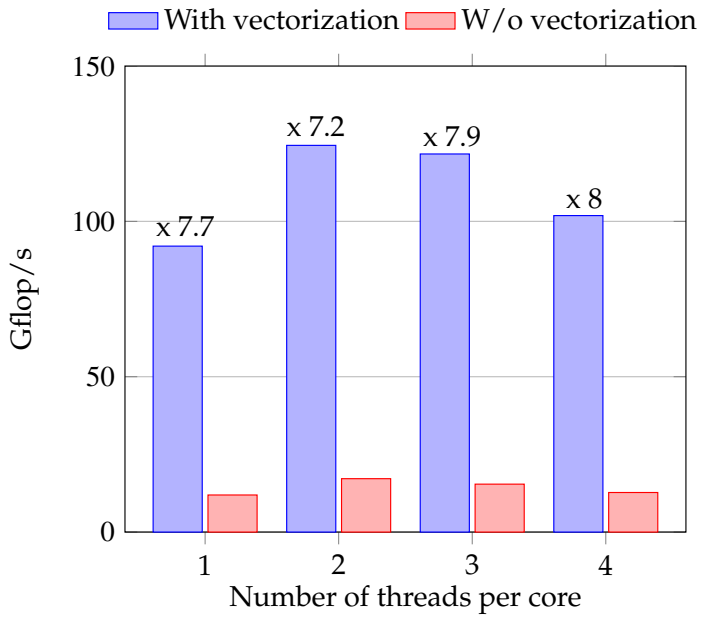


Figure 5.10 Vectorization impact on the TTI implementation on Intel Knights Corner.

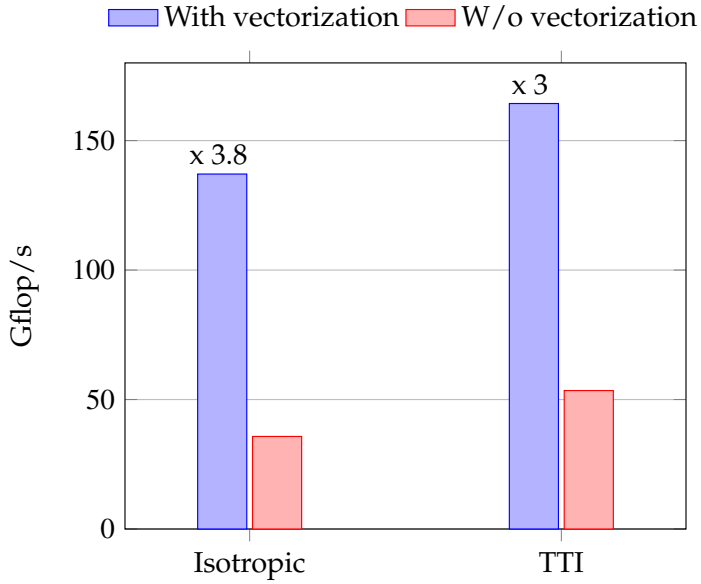


Figure 5.11 Vectorization impact on isotropic and TTI implementations on Sandy Bridge.

computation back and forth between the coprocessor and the host.

The openMP regions we intend to accelerate are offloaded to the card and in the case of the FDTD applications, we transfer the whole wavefield at each time step to perform the computation on the MIC. This results in latencies due to the utilization of the PCIe. The code in figure 5.12 explains the offload pragma used in the isotropic implementation.

```
1  !dir$ offload target(mic)                                    &
2  !dir$ inout(u0,u1)                                           &
3  !dir$ in(roc2,coef0,coefx,coefy,coefz)
4
5  !$omp parallel                                               &
6  !$omp shared(u0,u1,roc2)                                     &
7  !$omp shared(coef0,coefx,coefy,coefz)                        &
8  !$omp shared(xmin,xmax,ymin,ymax,zmin,zmax)                  &
9  !$omp private(lap,i,j,k)
10 do k=zmin,zmax
11     do j=ymin,ymax
12         !dec$ simd
13         do i=xmin,xmax
14             lap = coef0*u0(i,j,k)                            &
15                 + coefx(1)*(u0(i+1,j,k)                      &
16                 + u0(i-1,j,k))
17         end do
18     end do
19 end do
```

Figure 5.12 The offload pragma defines the targeted coprocessor and the data needed for the computation. The arrays $u0$ and $u1$ are transferred back and forth while the other arrays are copied only once.

Considering the Amdahl's law and the amount of time needed to perform transfers through the PCIe, we think that the offload mode is not likely to greatly enhance the performance of the FDTD applications. We need to conjointly use the host and the coprocessor while minimizing the effect of the PCIe transfers on the whole performance. This leads us to the symmetric mode which is presented in more details in the next section.

5.2.1.3 Symmetric Implementation

The symmetric mode is a hybrid and heterogeneous implementation. It involves the host and the MIC and consists in a hybrid MPI plus OpenMP implementation of the FDTD kernels. On a node level, we have one MPI rank per device i.e. 2 MPI ranks for a Sandy Bridge and a coprocessor. The MPI buffers involved during these communications correspond to the layers of ghost cells. Their number is equal to the half stencil order. Our domain decomposition is done along the Z direction because the slices sent in this case are contiguous in memory. We illustrate this MPI decomposition on figure 5.13.

As we are using a heterogeneous machine where the frequency and the number of the processing units are different on host and on MIC, we pay attention to our domain decomposition as load imbalance can induce a drop in performance. In this implementation, we have a static load balancing using a parameter denoted as `z-cut` equal to the ratio of the sizes of the sub-domains on Z direction respectively for the CPU and the MIC. For every run, we report timings for computation, MPI communications and overhead due to synchronizations. Figure 5.14 illus-

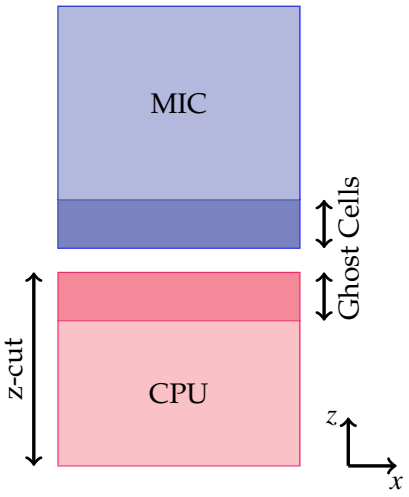


Figure 5.13 Domain decomposition for the symmetric implementation is only done on the Z direction. Only ghost cells are transferred through the PCIe.

trates percentages of these timings relatively to the main loop in FDTD for many values of `z-cut`. It depicts that for equal sub-domain sizes (`z-cut=1`), 5.3% of the time is wasted due to imbalance. We reduce this value to 1% of the execution time for a `z-cut` equal to 0.9.

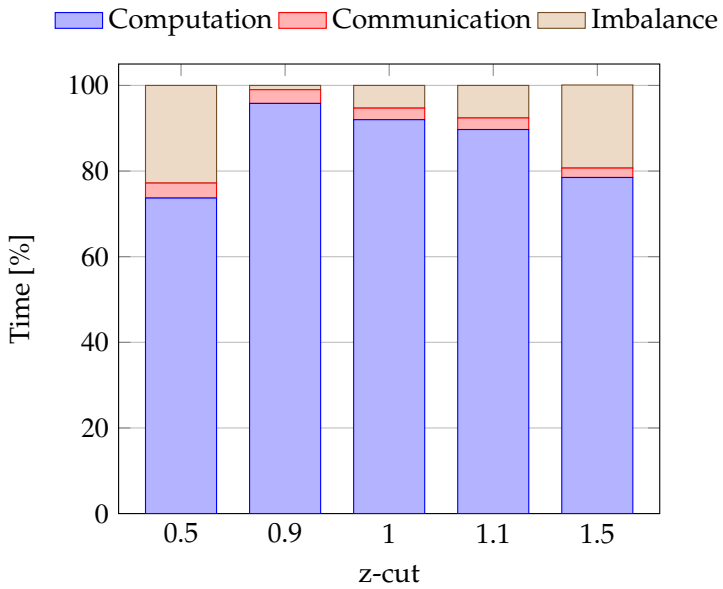


Figure 5.14 Percentages of the computation, the MPI communications and the overhead,relatively to the time spent in the main loop of the isotropic implementation for 2 values of z-cut.

As mentioned previously, we only need to transfer p layers of ghost cells where p is the half stencil order at each time step. As a consequence, we reduce the overhead due to data transfers on the PCIe compared to the offload mode where the

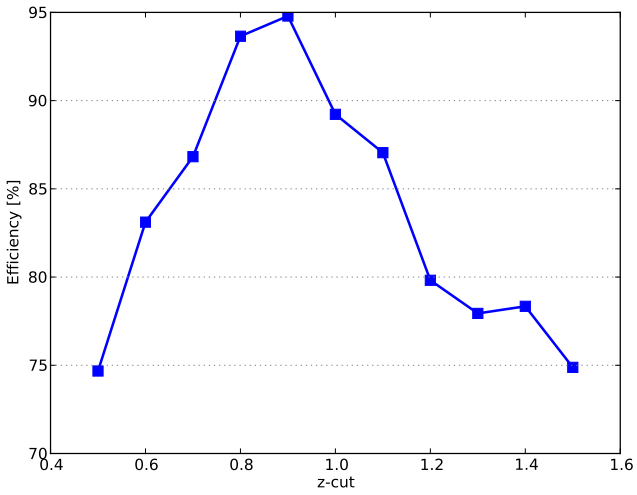


Figure 5.15 The efficiency corresponds to the ratio of the performance of the sub-domains on MIC and on CPU and the performance of the symmetric implementation. Varying the z-cut value enables the modification of the size of these sub-domains on both devices.

whole grid is transferred at each time step. For computation, we use the OpenMP threads. We deploy different number of threads on the CPU and on the MIC. We also pay attention to the cache block sizes we choose on these two different parts of the machine.

We compute an efficiency ratio defined as:

$$Efficiency = \frac{P_{symmetric}}{P_{native}(mic) + P(snb)}$$

The figure 5.15 plots the values of the efficiency for the same `z-cut` values in figure 5.14. The most *efficient* domain decomposition (98%) is one the that uses `z-cut=0.9` which confirms the results on figure 5.14.

Figure 5.16 illustrates the relative performance compared to a Sandy Bridge socket. We notice that that MIC is slightly better than the 2-socket Sandy Bridge implementation and that we have a 2.12x speedup when we the symmetric implementation.

In this chapter, we highlighted the impact of manycore architecture on FDTD. The increasing parallelism required to pay attention to the vectorization impact and to data movements. Heterogeneity required a tuning of the domain decomposition in order to reduce the load imbalance between the two devices due to the difference in compute capabilities. We have also constraints on the dimension where we apply the domain decomposition. We choose the Z direction in order to have contiguous data transferred via the PCIe interconnect.

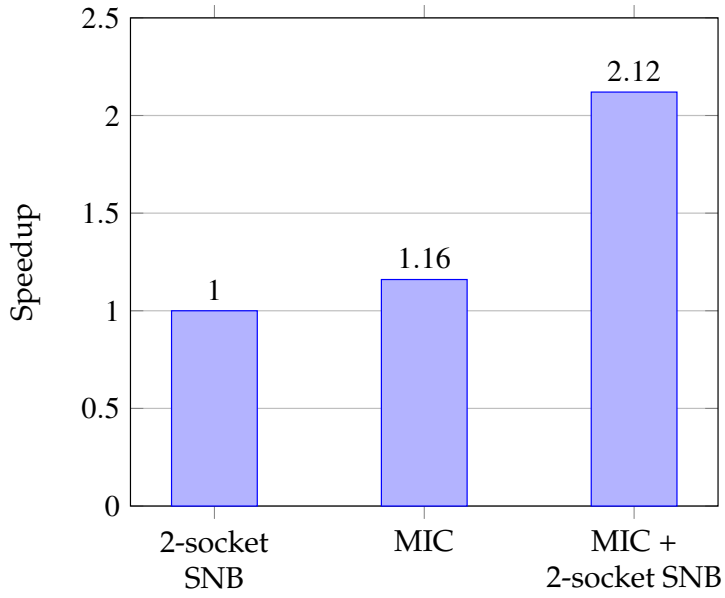


Figure 5.16 Relative performance compared to a single Sandy Bridge socket.

CHAPTER 6

Reverse Time Migration on Large Scale Systems

As described in chapter 1, Reverse Time Migration is one of the most important seismic imaging applications. It is widely used by the Oil and Gas companies since it makes a good trade-off between the quality of the image and the time-to-solution. Therefore, studying this application and characterizing it is important in order to prepare it for new architectures and for the upcoming Exascale systems.

We aim to conduct a co-design study applied to the Reverse Time Migration in isotropic and anisotropic media. Performance models for communication and computation will be used for the extrapolation to Exascale machines.

As an intermediate system, we use a cluster of Intel's Knight Corer (KNC). Porting RTM on such system will help us highlight the impact some of the Exascale constraints on it, such as heterogeneity, concurrency and memory limitations.

We study the impact of communications on the whole applications when we use a hybrid and heterogeneous implementation. We also consider the I/O issues since RTM is data intensive and we expect it to be limited by the I/O operations on Exascale systems.

6.1 Related Work on Reverse Time Migration

The section 1.2.1 of the first chapter 1 described from a high level point of view the main steps of RTM. The implementation we consider is a time-domain approximation of the partial differential equation governing the wave motion.

Algorithms 3 describes schematically the three steps required in RTM. We have 2 time loops, the first one is needed for the forward propagation during the recording time. Snapshots of the synthetic data are stored during the first loop. The second loop is the retro-propagation in time of the wave using the recorded data by the receivers. In order to perform the cross-correlation the forward values are restored at each time step.

Algorithm 3 Reverse Time Migration (RTM)

```
for t ← T_init, T_final do
    Forward_one_step(velocity_model)
    Save_snapshot(t)
end for
for t ← T_final, T_init do
    Backward_one_step(receivers)
    Restore_snapshot(t)
    Imaging_condition(Forward(t), Backward(t))
end for
```

6.1.1 State-of-the-art Implementations

RTM has always been subject to study on new architectures in order to take advantage of the new hardware features. Several adaptations of RTM can be found but we only present two of them. We start with a homogeneous implementation on CPUs. We present then the GPUs and their impact on the application.

Central Processing Unit (CPU)

RTM took advantage of the evolution of CPUs. Thanks to the increase of frequencies, the use of wide vectors and the instruction level parallelism (ILP), more complex computations were performed. The evolution of the memory hierarchies and the use of mechanisms such as the hardware and software prefetching, limitations due to the bandwidth can be overcome.

These new hardware features require modification of the implementation in order to ensure the vectorization and the efficient use of caches for example.

With the multi-core era, we have to consider the NUMA effect and make sure that the threads are correctly binded in order to avoid unnecessary traffic that hinder the overall performance.

After the node level optimizations, one should also consider the impact of the communications on performance and avoid as much as possible pending requests. We can consider load balancing between MPI processes through work stealing mechanisms which enables dynamic balancing.

Some CPU implementations rely on high speed networks to hide latencies when communicating like in Perrone et al., 2012. Their counter intuitive implementation took advantage of the low network latency and over decomposed the computational domain in order to save the snapshots in the node's main memory.

Graphics Processing unit (GPU)

Figure 6.1 is a high level description of a GPU architecture. It contains a bunch of multiprocessors. Each of them is composed of streaming threads. Multiprocessors have a user programmable memory shared by all thread blocks. For every

thread block we have a set of registers and a shared memory. They are equivalent to caches in CPUs.

GPUs are connected to a host via a PCIe to enable the data transfer between the devices. Accelerated parts of the code are fetched to the accelerator. GPUs are highly parallel architectures due to the streaming threads within multiprocessors. They are well suited for applications exploiting Single Instructions Multiple Data (SIMD) mechanism. The multi-threading in GPUs helps hiding memory latencies.

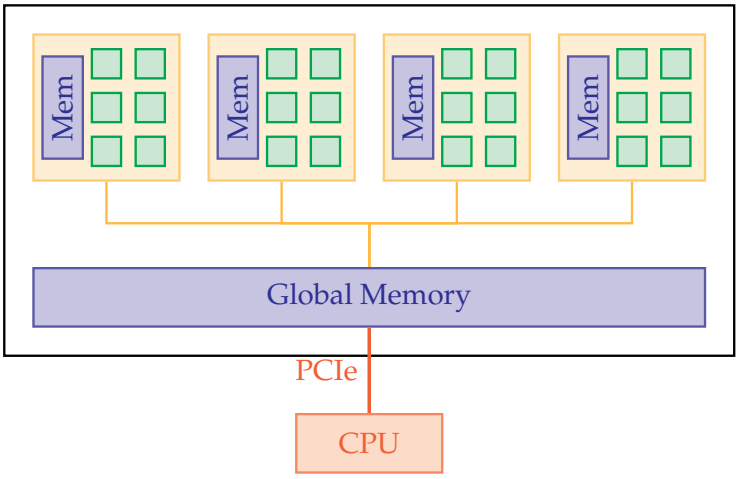


Figure 6.1 High-level description of GPUs architecture.

The main issues in GPUs are the memory limitations since global memory is not large enough to hold wavefields and the PCIe link that can be a bottleneck when computing the cross-correlation.

Another issue in GPU is the need to re-write the code using CUDA in order to ensure the optimal use of the architecture capacities.

Multi-GPU implementations like described in Micikevicius, 2009 is a solution to the shortage of memory on accelerators.

State-of-the-art implementations of RTM can be found in [Foltinek et al., 2009; Araya-Polo et al., 2011; Liu et al., 2009; Ortigosa et al., 2008] to name few. The challenges in these implementations is to overcome the little memory space available and the overhead due to data transfers trough the PCIe.

6.1.2 Velocity Models

A variety of benchmarks are used as input velocity models for RTM. Figure 6.2 is the velocity model for the 2004 BP benchmark. The study of sub-salt environments is highlighted since they are challenging for imaging applications due to the reflectivity at their edges. These salt structures can be encountered in the Gulf of Mexico and the off-shore West Africa [Billette et al., 2005].

The Marmousi benchmark in figure 6.3 is another velocity model representing a complex subsurface [Bourgeois et al., 1991; Versteeg, 1993].

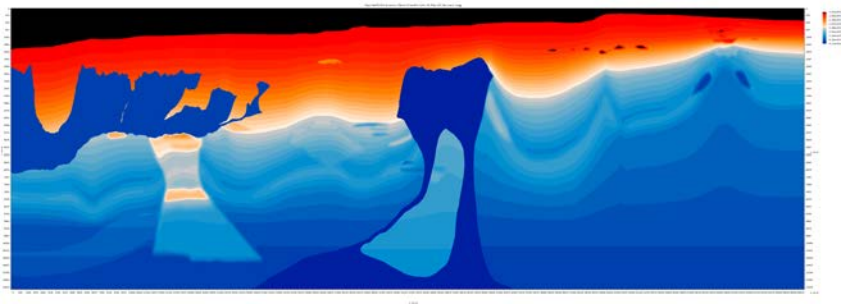


Figure 6.2 The 2004 BP velocity-analysis benchmark.

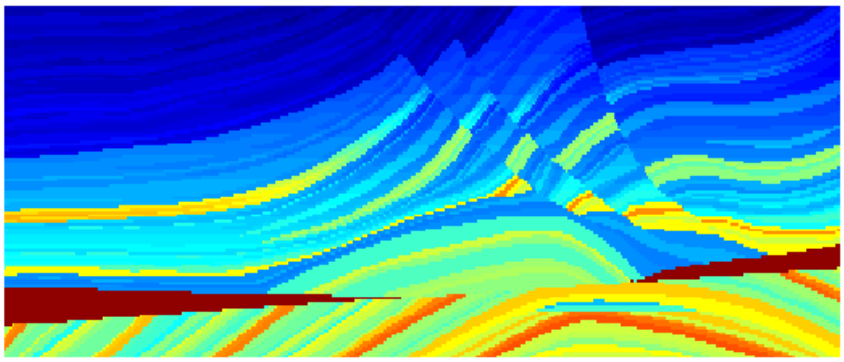


Figure 6.3 The Marmousi velocity model.

6.1.3 Snapshots and I/O Strategies

Cross-correlation, described in 6.1, requires an access to the value of the wavefield computed during the forward propagation for the same time step. This means that we need to save these values for each time step which results in tremendous usage of memory space. Domain decomposition techniques may reveal insufficient to fulfill the storage requirements and the network latencies can result in performance degradation because of the increase of the communication over computation ratio [Anderson et al., 2012].

Figure 6.4 is an illustration of the temporal cross-correlation in Reverse Time Migration. We present the forward propagation from the initial time step until the end of recording. We then have a propagation backward in time. The green circles in the middle correspond the imaging condition.

$$i(x,y,z) = \sum_{shots} \sum_{t=0}^{t_{max}} s(x,y,z,t) r(x,y,z,t) \tag{6.1}$$

We present the main computational strategies used in adjoint state problems. The aim of these strategies is to reduce the burden of the disk I/O while maintaining a minimum impact on the resulting image. Dussaud et al., 2008 describes these approaches and give also the complexities in terms of storage space and number of computations.

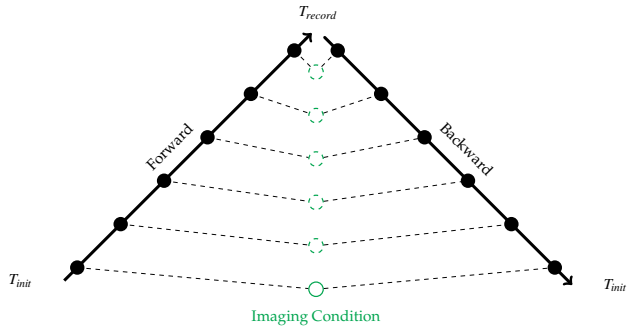


Figure 6.4 Computation of the imaging condition requires the forward and the backward wavefields at the same time step. Retrieving the necessary value of the forward field when retro-propagating the backward field can result in I/O bottlenecks.

Re-computation of the whole wavefield. At each time step in the backward propagation, we perform two wave propagations the velocity model and the recorded data and then compute the imaging condition for the current time step. We only need to store a single forward wavefield for the next computations.

In terms of computation, this method requires $\sum_{t=1}^{T}(T-t) = O(T^2)$ floating point operations. The memory usage is only $O(N)$.

Naive Wave field Storage. Forward wavefield values are stored for each time step. This approach requires prohibitive storage space and is not appropriate for industrial applications using real datasets.

Memory requirement is $\sum_{t=1}^{T} N = O(N.T)$ while computation for the forward propagation is equal to zero.

Wave field Storage. Forward wavefield values are stored every *k*-time step. During the backward propagation, the needed forward value is recomputed starting from the nearest *k*-time step. This approach reduces the amount of memory needed by a factor *k*. On the other hand, the amount of computation increases in the backward propagation. For this approach, we need to make a trade-off between the number of values stored and the number of computations performed in order to satisfy the imaging condition.

Computation $O(k.T)$ and memory $O(k.N)$.

Boundaries Storage. This approach concerns only the boundaries of the computational domain when damping is required to remove the artifacts due to reflections on the boundaries of the grid. During the forward propagation, only these values are saved. This approach reduces significantly the storage space needed even if it implies more computations.

Computation $O(d.N^{2/3})$ where d is the length of the damping zone and memory $O(N.T)$.

Checkpointing [Symes, 2007]. We store several pairs (u^{t-1}, u^t) during the forward step. When a boundary condition like PML is used, we store its value for the time step t. In order to recompute the value of the wavefield in the backward step, these pairs can be used as initial values. Optimal selection of these pair was introduced by Griewank et al., 2000.

Griewank requires $O(logN)$ computations.

Random Boundaries Strategies. Clapp, 2009 gives an implementation of random boundaries in Reverse Time Migration in order to avoid the I/O bottleneck encountered when the imaging condition is performed. In this approach reflections on boundaries are distorted in order to minimize artifacts.

Data compression can be used with all the previous methods in order to accelerate the I/O operations. This may result in the reduction of the amount of data stored and transferred but this can be done at the expense of the image quality. It is also important to have a low compute cost for the compression. Otherwise, overhead due to compression is subject to decrease performance.

6.2 Performance Modeling of RTM

$$T_{rtm} = \underbrace{T_{fwd} + T_{bwd} + T_{snap} + T_{ic}}_{T_{intranode}} + \underbrace{T_{comm}}_{T_{internode}} \tag{6.2}$$

Where

T_{fwd} Cost of FDTD applied to input models.

T_{bwd} Cost of FDTD applied to recorded data.

T_{snap} Computational cost of the snapshot strategy.

T_{ic} Cost of the imaging condition.

T_{comm} Cost of communications.

We make some assumptions in order to facilitate the modeling of the cost of each part of the algorithm. We first assume that communications are not overlapped by computations. Both their contributions are counted. We then consider single precision data and we don't use any compression methods for I/O operations.

6.2.1 Computation Costs

We use the notations as defined in table 6.1 for the following model.

$$T_{comp} = T_{fwd} + T_{bwd} + T_{snap} + T_{ic}$$

$N = n_x.n_y.n_z$	Domain size
$d_x,\ d_y,\ d_z$	Width of damping layers on x, y and z directions respectively
p	Half stencil order
T	Number of steps
c	Number of cores per node
th	Number of threads per core
f	Frequency
r	Number of FP ops per core
$\Phi = r.c.f$	Peak rate of FP ops per node
φ	Number of FP ops of the kernel
ζ	Number of FP ops per lattice update per time step
β_{mem}	Memory bandwidth
α_{net}	Network latency
β_{net}	Network bandwidth
R	Number of MPI ranks

Table 6.1 Notations used in the models of RTM.

Hypothesis

$$\begin{aligned} T_{fwd} &= T_{bwd} \\ &= \frac{T}{\Phi}\,\varphi_{kernel} \\ &= \frac{T\,N}{\Phi}\,\zeta_{kernel} \end{aligned}$$

Where $kernel \in \{iso, tti\}$ and ζ_{kernel} is defined as the number of floating point operations per lattice update per time step.

Similarly, we define T_{ic}.

$$\begin{aligned} T_{ic} &= \frac{T}{\Phi}\,\varphi_{ic} \\ &= \frac{T\,N}{\Phi}\,\zeta_{ic} \end{aligned}$$

We obtain:

$$\begin{aligned} T_{comp} &= \frac{3\,T\,N}{\Phi}\,\zeta_{kernel} + \frac{T\,N}{\Phi}\,\zeta_{ic} \\ &= (3\,\zeta_{kernel} + \zeta_{ic})\,\frac{T\,N}{\Phi} \end{aligned}$$

For imaging condition, we need to perform an addition and a multiplication which results in $\zeta_{ic} = 2$. According to the table 3.1, the number of floating point operations is equal to $\zeta_{iso} = 7\,p + 5$ for isotropic implementation and $\zeta_{tti} = 48\,p + 100$ for TTI.

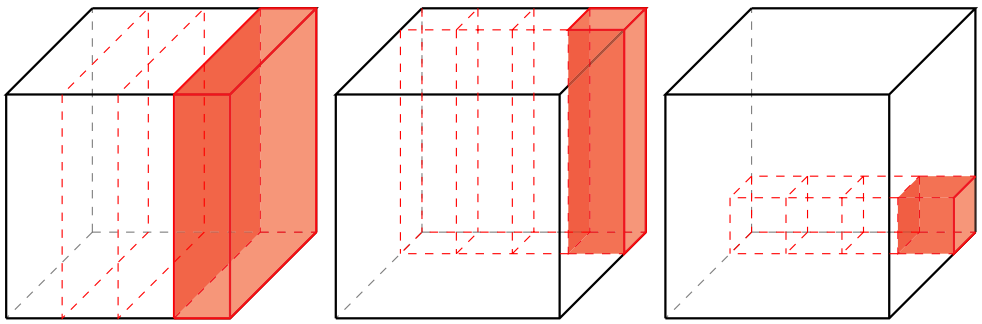
Finally, we obtain the computation time depicted in table 6.2 for isotropic and TTI implementations.

	Isotropic	*TTI*
T_{comp}	$7\ (p+1)\ \frac{T\ N}{\Phi}$	$(48\ p+102)\ \frac{T\ N}{\Phi}$

Table 6.2 Computation Time for isotropic and TTI kernels and imaging condition.

6.2.2 Communication Costs

Domain decomposition is very important since the number of elements transferred depends on the strategy we opt for. We consider the decomposition strategies illustrated in figure 6.5. We consider a cubic computational domain which implies $N = n^3$.



(a) Slab. **(b)** Pencil. **(c)** Cube.

Figure 6.5 Spatial Domain Decomposition

Each MPI rank communicates with its neighbors using point-to-point MPI calls. Communication time is proportional to the volume of data transferred and can be modeled using the following expression.

$$T_{comm} = \frac{4 \times h(n,R)}{\beta_{net}}$$

$h(n,R)$ designates the number of elements transferred per MPI rank. Its value depends on the domain decomposition strategy implemented. The size of the surfaces transferred are summarized in tabular 6.3.

	slabs	*pencils*	*cubes*
$h(n,R)$	$p\ n^2$	$\frac{p\ n^2}{\sqrt{R}}$	$\frac{p\ n^2}{R^{2/3}}$

Table 6.3 Size of the surfaces transferred depending on the domain decomposition method.

The figures 6.6 and 6.7 plot the execution time of the MPI calls for each MPI rank considering the domain decomposition methods presented above. These figures show that the slab subdomains imply higher communication cost compared to pencil and cubic subdomains.

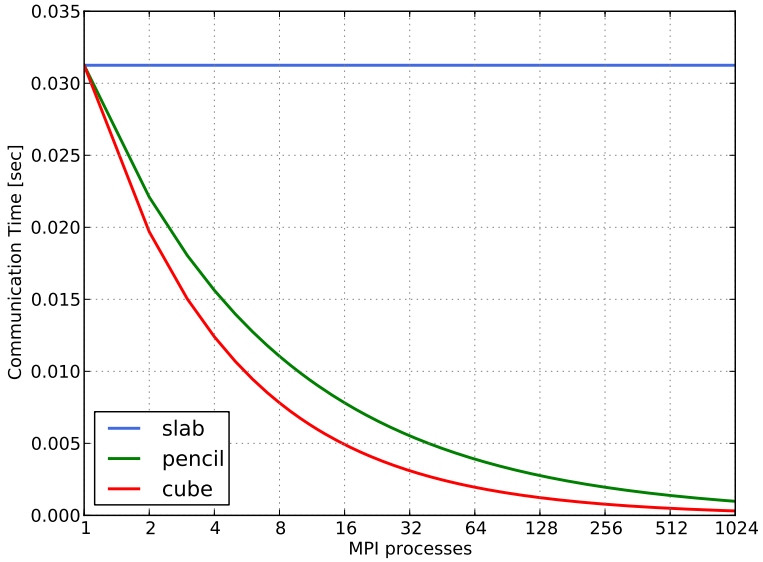


Figure 6.6 Strong Scaling. Communication time per MPI rank for fixed values of N, p and β_{net}.

These models are only considering the case of homogeneous cluster. All nodes have the same hardware characteristics. In the case of a heterogeneous cluster, containing accelerators such as GPUs or coprocessors like Intel's MIC, the communication model will be different since these devices are connected to the host via PCIe and as a consequence the bandwidth will be quite different from the network bandwidth.

We also assume no-overlapping of the communications by computations which means that execution time of the application is the sum of computation time by communication time as described by the expression 6.2.

6.2.3 Snapshot Strategy Costs

The implementation we consider for RTM has an isotropic and anisotropic versions. It uses a snapshot strategy based on the storage of boundaries. The storage is performed every nt time steps. The intermediate values are computed using an interpolation.

The number of snapshots is equal to

$$\lceil \frac{T-2}{nt} \rceil + 1 \tag{6.3}$$

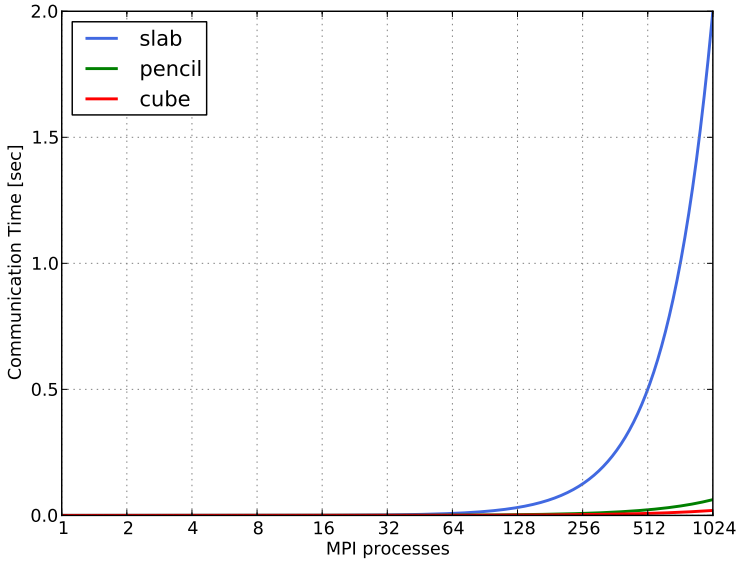


Figure 6.7 Weak Scaling. Communication time per MPI rank for fixed values of N, p and β_{net}. We use the same values as the strong scaling experiment.

The size of the boundaries stored for each time step is equal to

$$2d_z\ n_x\ n_y + 2d_y(n_x - 2d_x)(n_z - 2d_z) + 2d_x n_y(n_z - 2d_z) \tag{6.4}$$

We need to consider the number of the wavefields involved in order to have a precise estimation of the memory requirements in this implementation. In the isotropic case we have a single wavefield, while for TTI we need to multiply the previous formula by 2 since we have two wavefields p and q according to the equation 2.6.

Hypothesis: $d_x = d_y = d_z = d$ and $n_x = n_y = n_z = n$. This implies some simplifications in the equation 6.4 and gives us the size of the snapshots as a function of the size of the velocity model and the number of time steps.

$$2\ d\ (3\ n^2 - 6\ n\ d + 4\ d^2)\ (\lceil \frac{T-2}{nt} \rceil + 1) \tag{6.5}$$

Considering the simplifying hypothesis concerning the computing domain and the damping zone, we study the impact of the boundaries storage strategy with a downsampling factor on the size of the snapshots. Figure 6.8 illustrates the size of these snapshots as a function of the size of the velocity model. On the other hand, on figure 6.9 we show the correlation between time steps on the number and hence the size of time steps.

6.3 Implementation of RTM for Multi-node of Many-Core

This section is a proof of concept of porting an RTM code on cluster of KNCs. The main concern when it comes to time domain implementations is to handle the

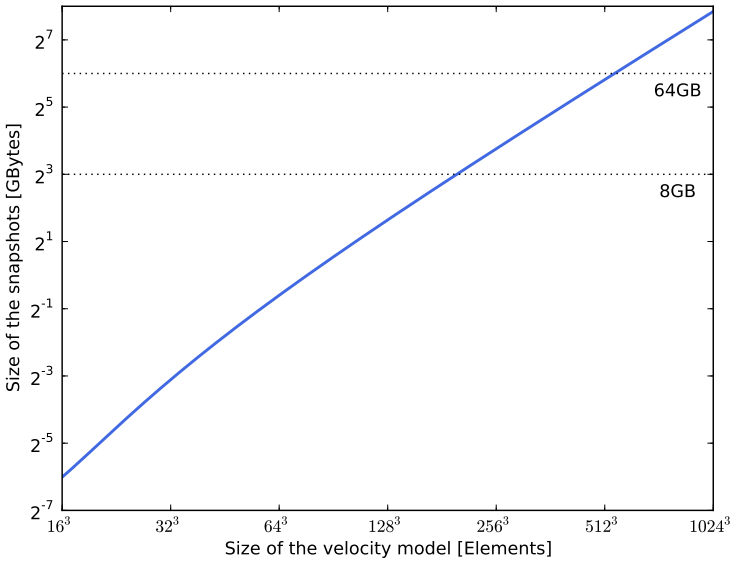


Figure 6.8 Size of snapshots as a function of the velocity model size. We give memory size on Sandy Bridge (64GB) and on MIC (8GB) as upper bounds on a KNC node.

I/Os efficiently.

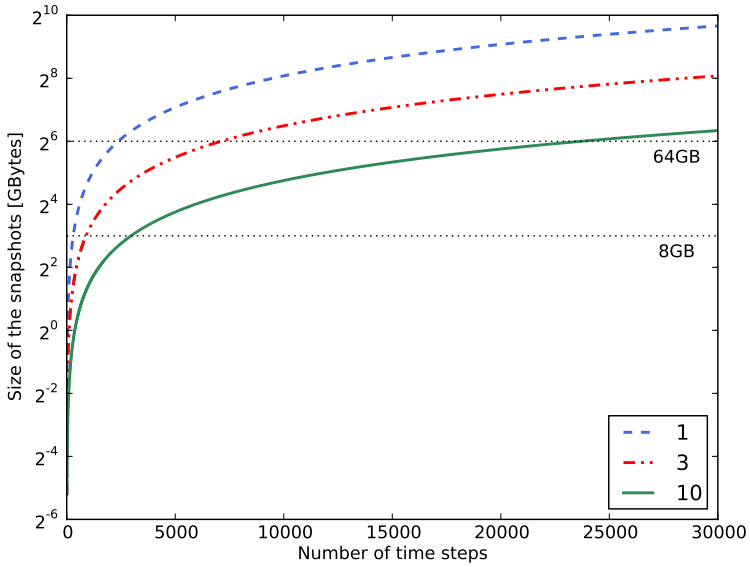
We consider a single shot for our study. Memory is going the main issue on the current version of MICs. On the KNC we used, the maximum memory we had access to is equal to 8GB. Snapshots generated for a single shot is counted in Tera bytes.

As a result, a single KNC with 2 coprocessors won't be able to hold one shot of RTM. Inevitably, we resort to use MPI and domain decomposition to get around the memory capacity shortage on MICs.

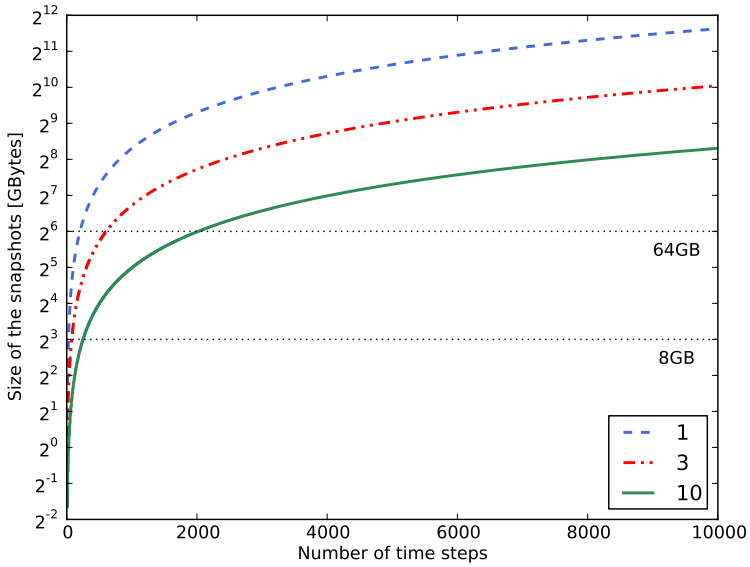
6.3.1 Test System

The target system used for our experiments is the machine Stampede from the Texas Advanced Computing Center (TACC) at the University of Texas at Austin. This machine is ranked 7 in the TOP500 list of June 2014 with a theoretical peak performance equal to 8.5 PFlop/s and maximum power usage equal to 4.5 MWatts.

Stampede has 6400 nodes, each one containing at least one Intel Xeon Phi co-processor. Only 480 nodes have 2 coprocessors. The interconnect is FDR InfiniBand delivering a network performance equal to 56 Gb/s to the node. The table 6.4 gives more details on the main hardware characteristics of Stampede. Compute nodes characteristics are summerized in table 6.5. We note that the memory available on host is only 32 GB compared to the 64 GB we have on the KNC prototype 5.1 used for the FDTD applications. We also note that on Stampede we have the Xeon Phi SE10P which has lower frequency and less memory than Xeon Phi 7120.



(a) A small velocity model 256^3 with a damping width equal to 22 on x, y and z directions.



(b) A real velocity model $621 \times 711 \times 1201$ with a damping width equal to 22 on x, y and z directions.

Figure 6.9 Size of snapshots as a function of the number of time steps which is determined using the CFL condition and the discretization steps. We consider a small and a real velocity models with 3 downsampling factors 1, 3 and 10. As we increase the downsampling factor, we reduce the size of the snapshots.

The software stack characteristics are shown on table 6.6. The Manycore Platform Software Stack (MPSS) available on Stampede when we performed the porting of RTM is the MPSS 2.1.

The compute nodes topology was captured using the version 1.8 of the Portable Hardware Locality (hwloc) tool. Figure 6.11 shows that the coprocessor `mic0` is attached to socket 0 along with the Gigabit Ethernet interfaces and the local disk `sda` while the coprocessor `mic1` shares the PCI link to the socket 1 with InfiniBand (IB) card. This topology is not symmetric relatively to the IB card. Notice also that on nodes with single coprocessor like the figure 6.10 the coprocessor `mic0` is linked to the socket 1 and not to socket 0.

Nodes	2 8-core XeonE5 processors 1 or 2 61-core Xeon Phi coprocessor	6400 nodes
Memory	32 GB / node	206 TB (aggregate)
Shared disk	Lustre 2.1.3 parallel file system	14 PB
Local disk	SATA (250 GB)	1.6 PB (aggregate)
Interconnect	InfiniBand Mellanox Switches / HCAs	FDR 56 Gb/s

Table 6.4 Characteristics of Stampede used to run hybrid version of RTM. Courtesy of TACC

Component	**Technology**
Sockets per Node/Cores per Socket	2/8 Xeon E5-2680 2.7GHz
Coprocessors/Cores	1/61 Xeon Phi SE10P 1.1GHz
Motherboard	Dell C8220, Intel PQI, C610 Chipset
Memory Per Host	32GB 8x4G 4 channels DDR3-1600MHz
Memory per Coprocessor	8GB GDDR5
Interconnect	
Processor-Processor	QPI 8.0 GT/s
Processor-Coprocessor	PCI-e
PCI Express Processor	x40 lanes, Gen 3
PCI Express Coprocessor	x16 lanes, Gen 2 (extended)
250GB Disk	7.5K RPM SATA

Table 6.5 Characteristics of compute node in Stampede. Courtesy of TACC.

Component	**Version**
Kernel	2.6.32-358.18.1.el6.x86_64
Manycore Platform Software Stack (MPSS)	2.1
Compiler	Intel 13.1.0
MPI	Intel 4.1.0.030

Table 6.6 Software stack on Stampede.

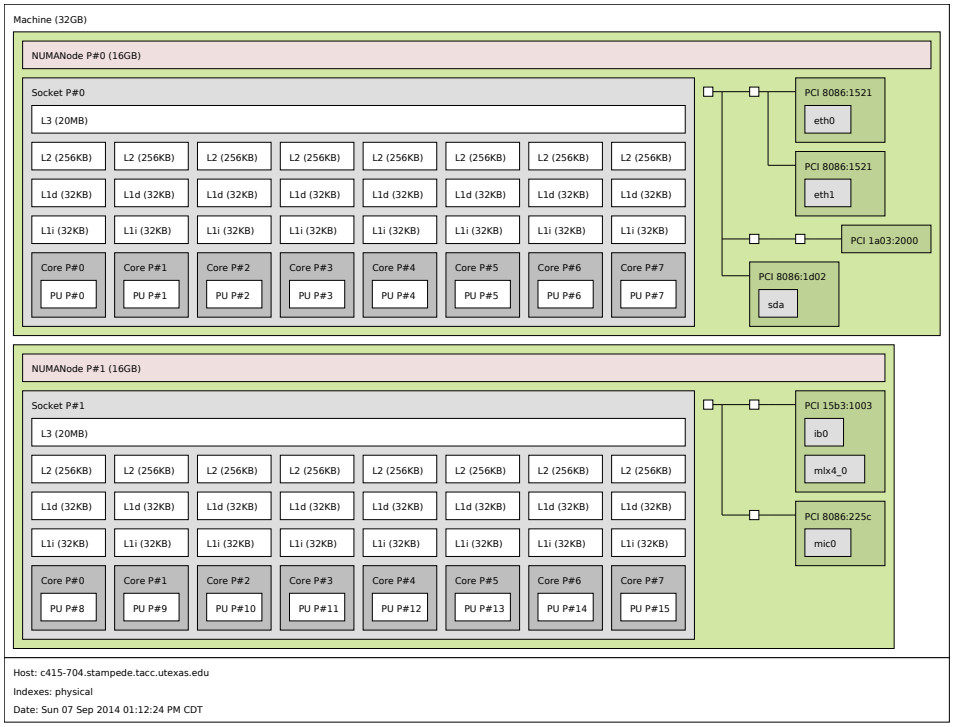


Figure 6.10 Topology of a compute node on Stampede with a single coprocessor.

We performed bandwidth measurements on Stampede using the Intel MPI Benchmark (IMB) tool. Figures 6.12 and 6.13 depicts the bandwidth values and communication time respectively. We give values of unidirectional bandwidth using pingpong benchmark for intra-node and inter-node communications. In comparison with the measurement that we performed on the Intel Xeon 7140 5.1, the bandwidth values for the intra-node communications on figure 5.3 are higher that the values that we measure on Stampede. Notice that the MPSS version available on Stampede and the MPI used are older.

6.3.2 RTM Implementations

We study a TTI implementation of RTM and we use a velocity model of size 512^3. We consider 3 versions of this implementation of RTM. First we have an MPI only implementation. We have a hybrid version where we use MPI for communications and OpenMP to parallelize computations within the node. This hybrid version was run on Sandy bridge nodes only and in a symmetric mode using the KNCs as presented in the previous chapter 5.2.1.3.

Figure 6.14 illustrates the distribution of the MPI ranks and the OpenMP threads within the nodes of the cluster Stampede. We did not perform any work load balancing on the current version of RTM. We have the same sub-domain per device. We only adapt the value of runtime variable and the cache size.

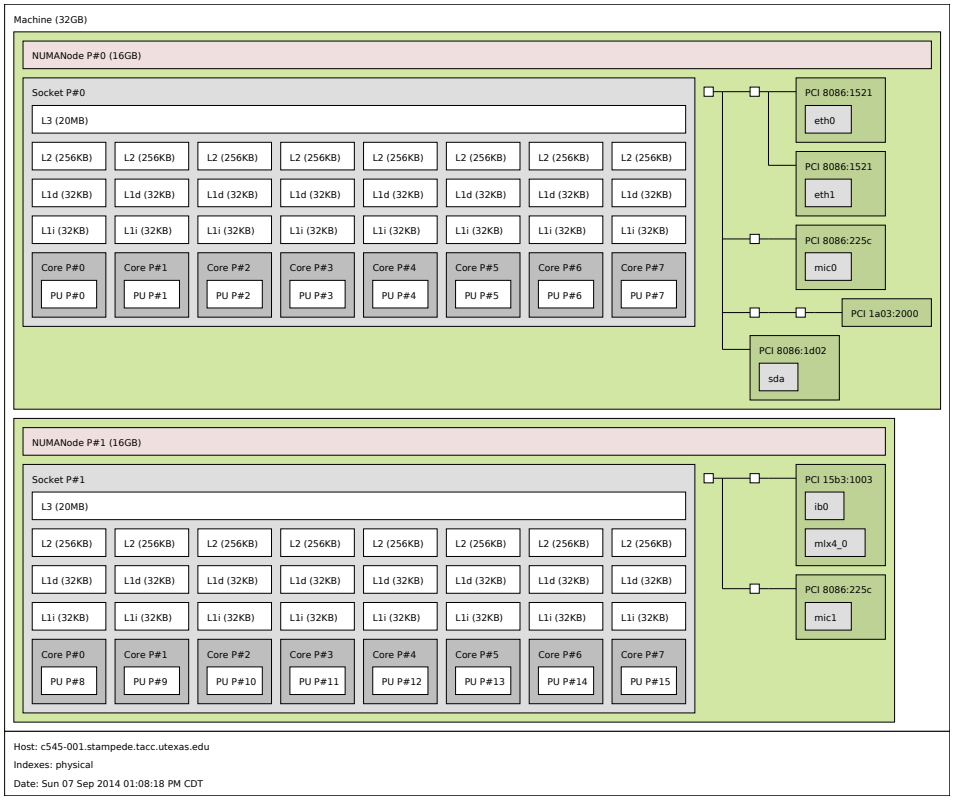
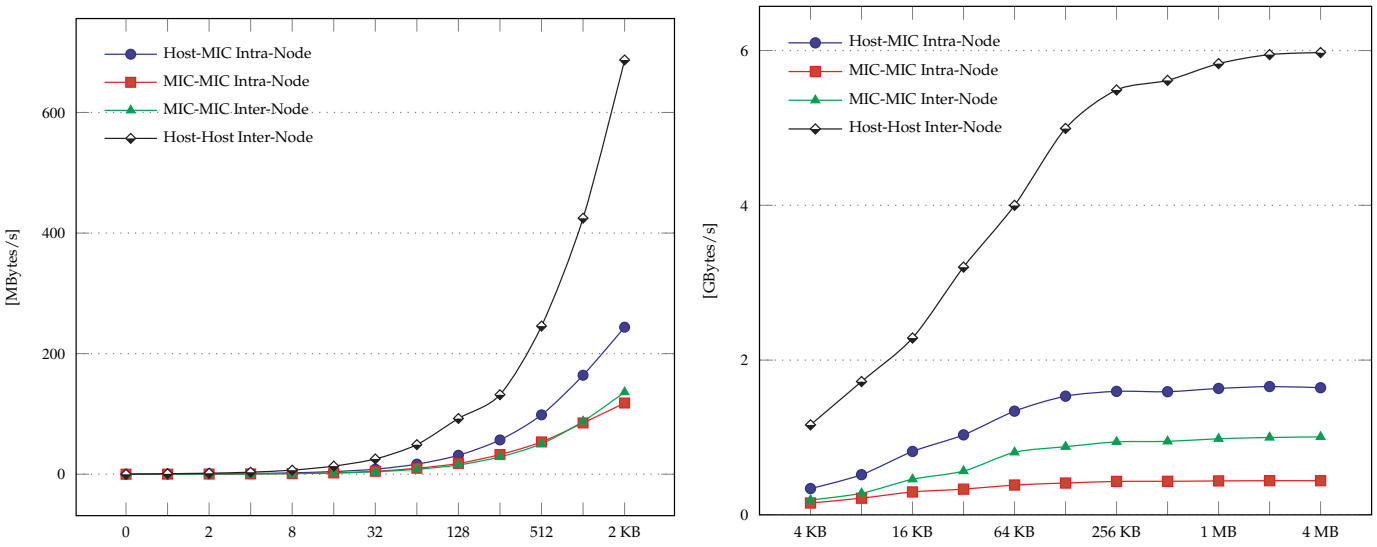


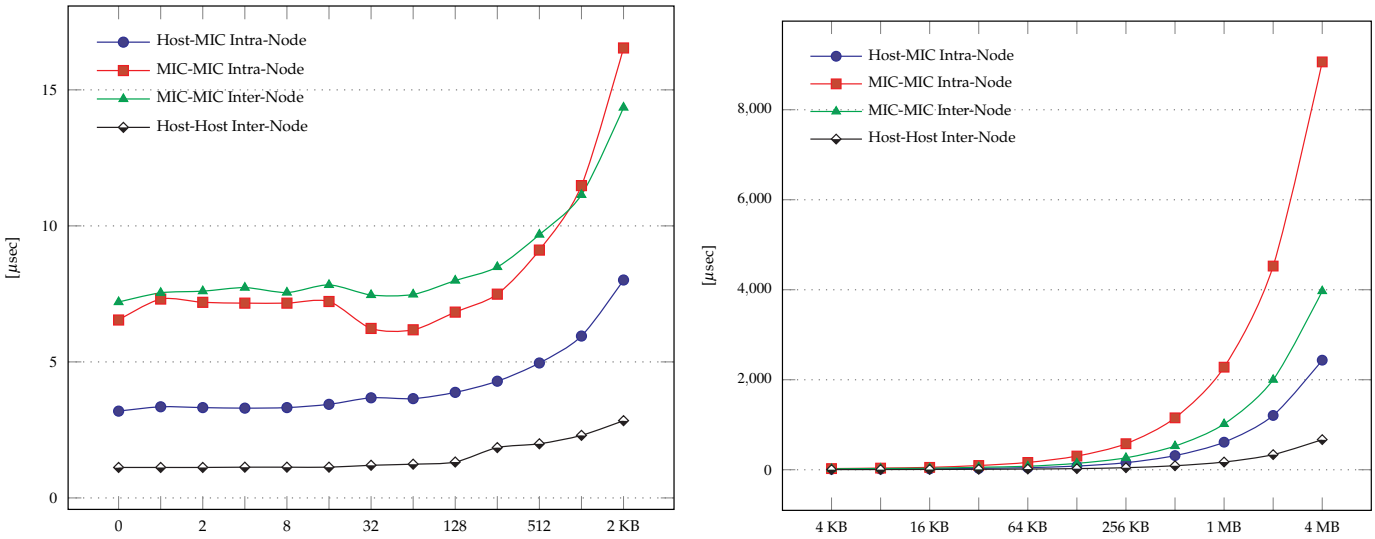
Figure 6.11 Topology of a compute node on Stampede with 2 coprocessors.



(a) Small messages

(b) Large messages

Figure 6.12 Bandwidth measurements on Stampede for intra- and inter-node communications.



(a) Small messages

(b) Large messages

Figure 6.13 Communication time for inter- and intra-node communications

We perform strong scalability study using the velocity model of size 512^3. We plot the execution time in two cases : a single coprocessor per node and two coprocessors per node. Figure 6.16 illustrates these timing for the whole shot, MPI communications, computation and I/Os. As we increase the number of nodes, the execution time is bounded by the communication overhead. The use of 2 coprocessors does not seem to improve performance. We believe that this is due to the poor intra-node bandwidth values that we reported on figure 6.12.

We used the Intel Trace Analyzer and Collector (ITAC) to report on the MPI communications. We could not run it on Stampede since the tool was not available. We only report the MPI trace for a single node execution using 2 coprocessors in figure 6.15. Blue regions refer to computations while the red ones correspond to MPI communications. P0 corresponds to the host and P1 and P2 refer to `mic0` and `mic1` respectively. We notice strong disparities between the host and the coprocessor. This suggests that we need a more adapted domain decomposition as we done for the FDTD applications. For RTM, dynamic load balancing is more adequate for large systems. Static load balancing using a manual domain decomposition in unlikely to be efficient for such applications.

Figure 6.17 gives a comparison between all the implementations considered for these experiments. We give the execution time as well as the speedup relatively to an execution on a single node. As we can see, the results for the hybrid version on Sandy Bridge give the best results for the first 64 nodes followed by the MPI only version then the symmetric version. In terms of scalability, the MPI only version is the version that scales the best. This does not reflect the real execution time. On Stampede using 2 programming models with RTM showed good results. But the

use of coprocessors did not improve execution time. Load balancing issues, high latencies and low bandwidth degrade the overall performance.

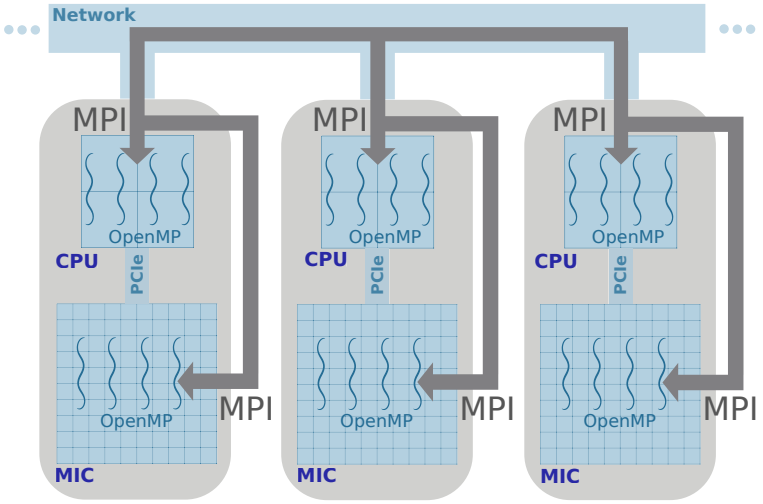


Figure 6.14 Hybrid and heterogeneous implementation of RTM on a KNC cluster. Communications are performed by MPI processes while computations are made by OpenMP threads.

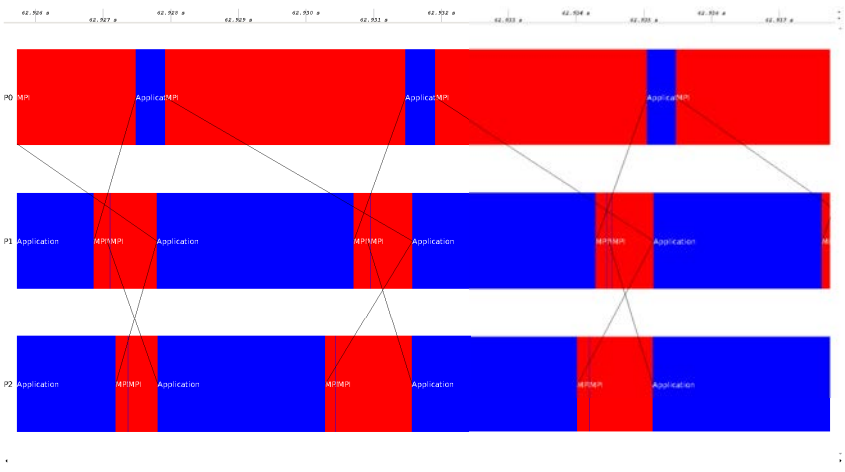
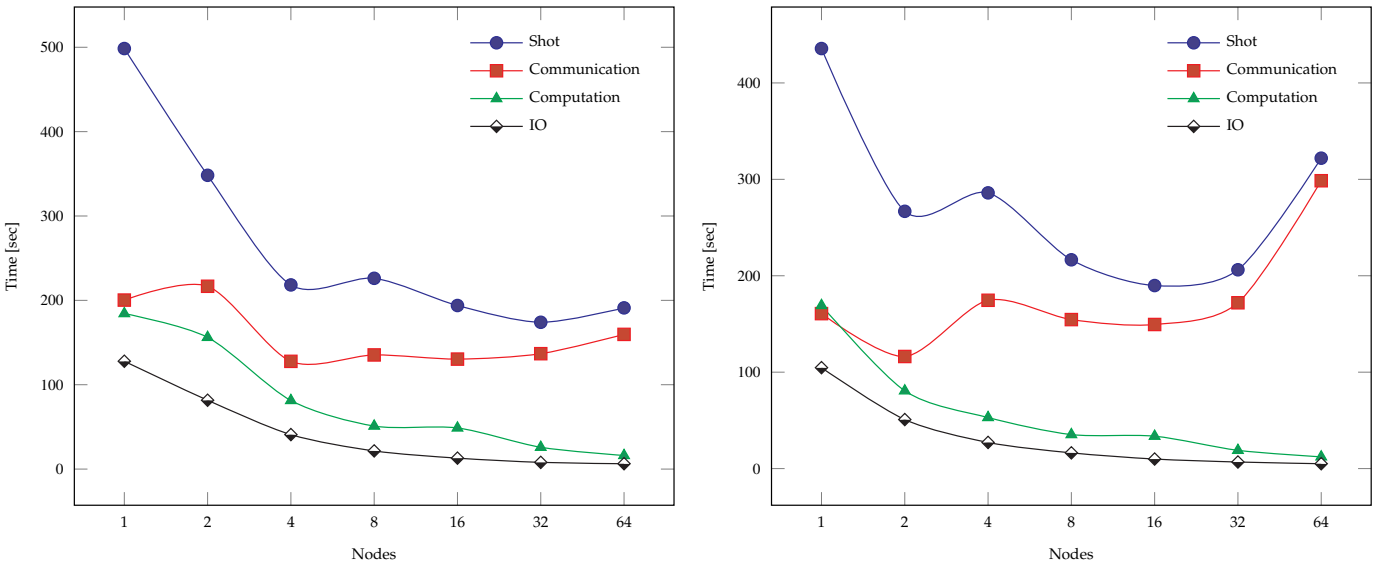
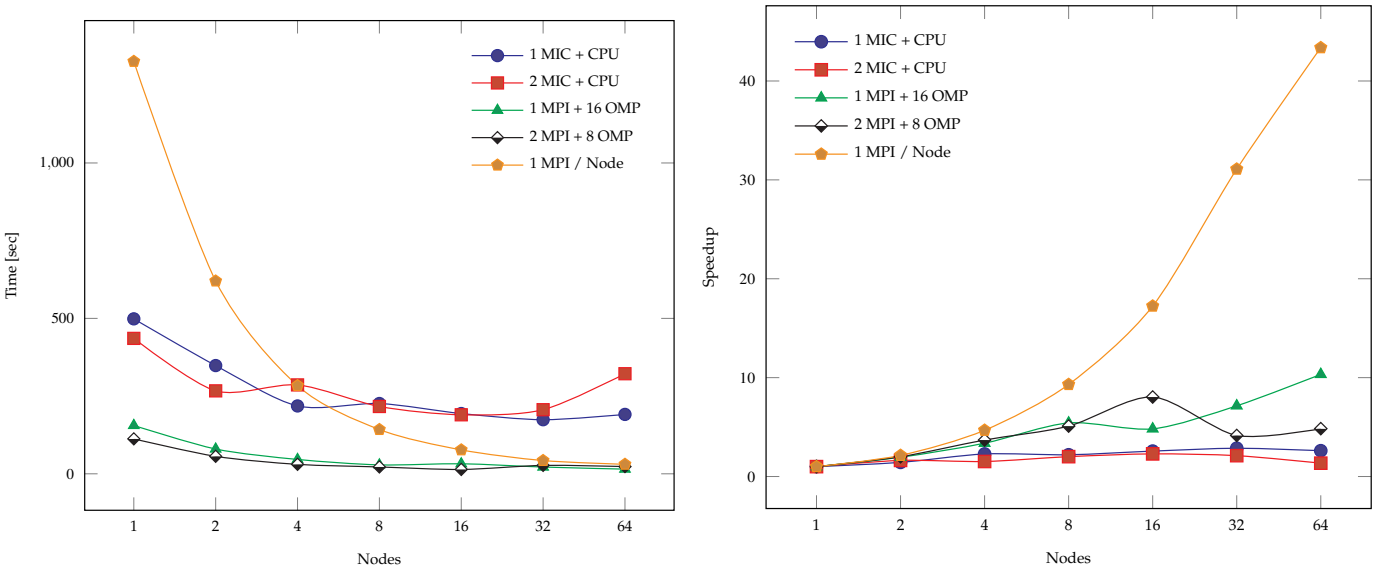


Figure 6.15 MPI communications of RTM in intra-node.



(a) 1 MIC + CPU **(b)** 2 MIC + CPU

Figure 6.16 Performance of RTM in terms of communication, computation and IO on Stampede.



(a) Execution times **(b)** Speedup

Figure 6.17 Comparison of the different implementations of RTM on Stampede.

Conclusion and Future Work

This work analyzes extensively the seismic imaging application Reverse Time Migration (RTM) and covers all the aspects that influence its performance. As a preparation for the Exascale systems, we highlight the features that impact performance and we consider for this the hardware and the software parts.

One of the major challenges when it comes to optimizing an application is to determine the values of the optimization parameters and their optimal combination. The sampling-based tool ASK permitted us to determine this combination for the FDTD kernels.

We started by studying the compute core of this application consisting in a finite-difference time-domain (FDTD) kernel. In order to characterize these kernels, we used the Roofline model as a simple and straightforward model to illustrate the interaction between the application and the underlying architecture. This characterization showed that FDTD kernels are bandwidth limited and that they are greatly impacted by the DRAM bandwidth. Hence, the applied optimizations targeted the data reuse and data access patterns. These optimizations resulted in increasing the arithmetic intensity of the application and approaching consequently the theoretical arithmetic intensity where we assume a perfect use of the last level cache. We also modeled the DRAM traffic with and without cache blocking and we based our model on the reuse distance histogram. The predicted values are pretty close to the real ones which constitutes a useful tool to predict the DRAM traffic on unavailable architectures.

The second step of our study consisted in porting the FDTD kernel on a many-core architecture such as the Intel Knights Corner (KNC) co-processor in order to highlight the influence of concurrency and heterogeneity. The use of the co-processor in native mode emphasizes the importance of vectorization and the optimal data reuse. Multi-threading in this particular architecture has an important impact on performance compared to a single thread per core but it can also hinder optimal data reuse. The simultaneous use of the host and the co-processor highlighted the impact of heterogeneity. We implemented an efficient symmetric version of the FDTD using these two devices. We use MPI and OpenMP programming models and we tune the domain decomposition in order to reduce the load imbalance due to compute capacity differences.

We conducted a feasibility study of RTM on large scale systems where we modeled the costs of computation, communication and I/O. We highlighted the impact

of domain decomposition and checkpointing strategy on this data intensive application. We then focused on heterogeneous systems involving manycore architectures like Intel Knights Corner Xeon Phi (KNC). We determined the bottlenecks that we are going to deal with on these particular architectures using performance models and showed that due to the small memory available on KNC, we are forced to over-decompose the computation grid over the nodes. We were also aware of the overhead introduced by the PCIe interconnect and its great impact on MPI communications. We ported a hybrid and heterogeneous implementation RTM on the system Stampede hosted in TACC. Our measurements confirmed the expected bottlenecks and showed the great impact of the MPI communications on performance. We also made a comparison between this hybrid and heterogeneous implementation with MPI only and hybrid implementations on multicore architectures. The current symmetric implementation of RTM struggles to efficiently scale due to the PCIe interconnect and due to poor memory hierarchy, in particular the inefficient shared last level cache (L2).

Future Work As heterogeneity did not impact positively the performance of RTM and the lack of memory capacity on the co-processor we used for our experiments resulted in an over-decomposition of the compute grid, we would like to study more in depth the impact of concurrency using the next generation of Intel's manycore architecture, the Knight Landing (KNL) since it is expected to address some of the limitations we encountered on the Knights Corner. The PCIe interconnect will be removed and the memory capacity and bandwidth will be increased considerably. We would like also to consider other manycore architectures such as the Kalray Multi-Purpose Processing Array (MPPA) architecture offering a CPU-like control units and floating point units. It also comprises a high speed low latency network on chip and limits power consumption to 5-10 Watt.

On the algorithmic side, we want also to pursue the study using other numerical schemes applied to RTM like the finite-element methods and more precisely the high-order Discontinuous Galerkin Method (DGM) since they are well suited to run on highly parallel architectures. Operators in DGM are applied locally and the high order of this method results in a high arithmetic intensity. We can also study an other approach of approximating the wave equation using an asymptotic method and more precisely the Kirchhoff method known to be computationally intensive which makes it a suitable candidate for manycore architectures. The Prestack Kirchhoff time migration (PSTM) is one of the most popular migration techniques thanks to its efficiency and simplicity.

Bibliography

Alkhalifah, T. (2000). "An acoustic wave equation for anisotropic media". In: *GEOPHYSICS* 65.4, pp. 1239–1250 (see p. 24).

Anderson, John E., Lijian Tan, and Don Wang (2012). "Time-reversal checkpointing methods for RTM and FWI". In: *Geophysics* 77.4, S93 (see p. 84).

Araya-Polo, Mauricio, Javier Cabezas, Mauricio Hanzich, Miquel Pericas, Felix Rubio, Isaac Gelado, Muhammad Shafiq, Enric Morancho, Nacho Navarro, Eduard Ayguade, et al. (2011). "Assessing accelerator-based HPC reverse time migration". In: *Parallel and Distributed Systems, IEEE Transactions on* 22.1, pp. 147–162 (see p. 83).

Ashby, Steve, P Beckman, J Chen, P Colella, B Collins, D Crawford, J Dongarra, D Kothe, R Lusk, P Messina, et al. (2010). "The opportunities and challenges of exascale computing". In: *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (November 2010)* (see p. 1).

Bader, Michael and Christoph Zenger (2006). "A cache oblivious algorithm for matrix multiplication based on peano's space filling curve". In: *Proceedings of the 6th international conference on Parallel Processing and Applied Mathematics*. PPAM'05. Poznań, Poland: Springer-Verlag, pp. 1042–1049 (see p. 48).

Barrett, R.F., S.D. Hammond, C.T. Vaughan, D.W. Doerfler, M.A. Heroux, J.P. Luitjens, and D. Roweth (2012). "Navigating an Evolutionary Fast Path to Exascale". In: *High Performance Computing, Networking Storage and Analysis, SC Companion:* 0, pp. 355–365 (see p. 9).

Batten, Christopher Francis (2010). "Simplified vector-thread architectures for flexible and efficient data-parallel accelerators". PhD thesis. Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science (see p. 7).

Beyls, Kristof and Erik D'Hollander (2001). "Reuse distance as a metric for cache behavior". In: *Proceedings of the IASTED Conference on Parallel and Distributed Computing and systems*. Vol. 14, pp. 350–360 (see p. 58).

Bhatele, Abhinav, Pritish Jetley, Hormozd Gahvari, Lukasz Wesolowski, W.D. Gropp, and L. Kalé (2011). "Architectural constraints to attain 1 Exaflop/s for three

scientific application classes". In: *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, pp. 80–91 (see p. 14).

Billette, Frederic and Sverre Brandsberg-Dahl (2005). "The 2004 BP velocity benchmark." In: *67th Annual Internat. Mtg., EAGE, Expanded Abstracts*. EAGE, B035 (see p. 18, 83).

Biondi, Biondo (2006). *3D seismic imaging*. 14. Society of Exploration Geophysicists (see p. 15, 27).

Borkar, Shekhar (2013). "Exascale computing - A fact or a fiction?" In: *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 3–3 (see p. 3).

Borkar, Shekhar and Andrew A. Chien (May 2011). "The future of microprocessors". In: *Commun. ACM* 54.5, pp. 67–77 (see p. 6).

Bourgeois, A, M Bourget, P Lailly, M Poulet, P Ricarte, and R Versteeg (1991). "Marmousi, model and data". In: *The Marmousi Experience*, pp. 5–16 (see p. 83).

Broquedis, Francois, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst (2010). "Hwloc: A Generic Framework for Managing Hardware Affinities in HPC Applications". In: *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. PDP '10. Washington, DC, USA: IEEE Computer Society, pp. 180–186 (see p. 69).

Butz, Arthur R. (1968). "Space filling curves and mathematical programming". In: *Information and Control* 12.4, pp. 314 –330 (see p. 47).

Camp, William J. and Philippe Thierry (2010). "Trends for high-performance scientific computing". In: *The Leading Edge* 29.1, pp. 44–47 (see p. 5, 6, 8, 12).

Carcione, J., G. Herman, and A. ten Kroode (2002). "Seismic modeling". In: *GEOPHYSICS* 67.4, pp. 1304–1325 (see p. 21, 23).

Carter, N.P., A. Agrawal, S. Borkar, R. Cledat, H. David, D. Dunning, J. Fryman, I. Ganev, R.A. Golliver, R. Knauerhase, R. Lethin, B. Meister, A.K. Mishra, W.R. Pinfold, J. Teller, J. Torrellas, N. Vasilache, G. Venkatesh, and J. Xu (2013). "Runnemede: An architecture for Ubiquitous High-Performance Computing". In: *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*, pp. 198–209 (see p. 3).

Christen, Matthias, Olaf Schenk, and Helmar Burkhart (May 2011). "PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures". In: *Proc. 25th IEEE International Symposium on Parallel and Distributed Processing (25th IPDPS'11)*. Anchorage, Alaska, USA: IEEE Computer Society, pp. 676–687 (see p. 47).

Chu, C. and P. Stoffa (2012). "Implicit finite-difference simulations of seismic wave propagation". In: *GEOPHYSICS* 77.2, T57–T67 (see p. 29).

Clapp, Robert G (2009). "Reverse time migration with random boundaries". In: *79th Annual International Meeting, SEG Expanded Abstracts*. Vol. 28, pp. 2809–2813 (see p. 86).

Coteus, Paul W., John U. Knickerbocker, Chung H. Lam, and Yurii A. Vlasov (2011). "Technologies for exascale systems". In: *IBM Journal of Research and Development* 55.5, p. 14 (see p. 5, 8).

Courant, R., K. Friedrichs, and H. Lewy (1967). "On the partial difference equations of mathematical physics". In: *IBM J. Res. Dev.* 11.2, pp. 215–234 (see p. 32).

Czechowski, Kenneth, Casey Battaglino, Chris McClanahan, Aparna Chandramowlishwaran, and Richard Vuduc (2011a). "Balance principles for algorithm-architecture co-design". In: *Proc. USENIX Wkshp. Hot Topics in Parallelism (HotPar)*. Berkeley, CA, USA (see p. 14).

Czechowski, Kenneth, Chris McClanahan, Casey Battaglino, Kartik Iyer, P.-K. Yeung, and Richard Vuduc (2011b). "Prospects for scalable 3D FFTs on heterogeneous exascale systems". In: *In Proc. ACM/IEEE Conf. Supercomputing (SC)*. (poster; extended version available as Georgia Tech report GT-CSE-11-02) (see p. 14).

— (2012). "On the communication complexity of 3D FFTs and its implications for exascale". In: *Proc. ACM Int'l. Conf. Supercomputing (ICS)*. San Servolo Island, Venice, Italy (see p. 14).

DARPA (2010). *Ubiquitous High Performance Computing (UHPC) program*. URL: `http://www.darpa.mil/Our_Work/MTO/Programs/Ubiquitous_High_Performance_Computing_(UHPC).aspx` (see p. 3).

Datta, Kaushik, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Kathy Yelick (Nov. 2008). "Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures". In: *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12 (see p. 43, 44).

Dave Turek (2009). "The Strategic Future: The Push to Exascale!" In: *IBM Science and Innovation Summit* (see p. 5).

Davis, Kristofer and Yaoguo Li (2011). "Fast solution of geophysical inversion using adaptive mesh, space-filling curves and wavelet compression". In: *Geophysical Journal International* 185.1, pp. 157–166 (see p. 47).

Ding, Chen and Yutao Zhong (May 2003). "Predicting Whole-program Locality Through Reuse Distance Analysis". In: *SIGPLAN Not.* 38.5, pp. 245–257 (see p. 59).

Dongarra, J. J. and A. J. van der Steen (May 2012). "High-performance computing systems: Status and outlook". In: *Acta Numerica* 21, pp. 379–474 (see p. 11).

Dongarra, Jack J, Piotr Luszczek, and Antoine Petitet (2003). "The LINPACK benchmark: past, present and future". In: *Concurrency and Computation: practice and experience* 15.9, pp. 803–820 (see p. 2).

Dongarra, Jack, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi, Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Zhong Jin, Yutaka Ishikawa, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias S. Mueller, Wolfgang E. Nagel, Hiroshi Nakashima, Michael E Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad Van Der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick (Feb. 2011). "The International Exascale Software Project roadmap". In: *Int. J. High Perform. Comput. Appl.* 25.1, pp. 3–60 (see p. 1, 9).

Dussaud, E, WW Symes, and P Williamson (2008). "Computational strategies for reverse-time migration". In: *78th SEG Annual Meeting*, pp. 2267–2271 (see p. 19, 84).

Eachempati, Deepak, Alan Richardson, Terrence Liao, Henri Calandra, and Barbara M. Chapman (2012). "A Coarray Fortran Implementation to Support Data-Intensive Application Development". In: *SC Companion*. IEEE Computer Society, pp. 771–776 (see p. 10).

Efron, B. and R. Tibshirani (1986). "Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy". In: *Statistical science* 1.1, pp. 54–75 (see p. 49).

Farmer, Paul A, Ian F Jones, Hongbo Zhou, Robert I Bloor, and Mike C Goodwin (2006). "Application of reverse time migration to complex imaging problems". In: *First Break* 24.9 (see p. 19).

Farmer, Paul, Zheng Zheng Zhou, and David Jones (2009). "The role of reverse time migration in imaging and model estimation". In: *The Leading Edge* 28.4, pp. 436–441 (see p. 18).

Fletcher, Robin P., Xiang Du, and Paul J. Fowler (2009). "Reverse time migration in tilted transversely isotropic (TTI) media". In: *Geophysics* 74.6, WCA179 (see p. 24).

Foltinek, Darren, Daniel Eaton, Jeff Mahovsky, Peyman Moghaddam, and Ray McGarry (2009). "Industrial-scale reverse time migration on gpu hardware". In: *2009 SEG Annual Meeting* (see p. 83).

Friedman, J.H. (2001). "Greedy function approximation: a gradient boosting machine.(English summary)". In: *Ann. Statist* 29.5, pp. 1189–1232 (see p. 50, 51).

Frigo, M., C.E. Leiserson, H. Prokop, and S. Ramachandran (1999). "Cache-oblivious algorithms". In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 285–297 (see p. 47).

Gahvari, Hormozd and William Gropp (2010). "An introductory exascale feasibility study for FFTs and multigrid". In: *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, pp. 1–9 (see p. 14).

Griewank, Andreas and Andrea Walther (Mar. 2000). "Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation". In: *ACM Transactions on Mathematical Software* 26.1, pp. 19–45 (see p. 86).

Hall, Mary, Richard Lethin, Keshav Pingali, Dan Quinlan, Vivek Sarkar, John Shalf, Robert Lucas, Katherine Yelick, Pavan Balaji ANL, Pedro C Diniz, et al. (2011). "ASCR Programming Challenges for Exascale Computing". In: (see p. 7).

Hoisie, Adolfy, Darren Kerbyson, Robert Lucas, Arun Rodrigues, John Shalf, Jeffrey Vetter, William Harrod, Sonia Sachs, Kevin Barker, Jim Belak, Greg Bronevetsky, Chris Carothers, Boyana Norris, and Sudhakar Yalamanchili (2012). *Report on the ASCR Workshop on Modeling and Simulation of Exascale Systems and Applications*. Tech. rep. (see p. 34).

Imbert, David, Khadija Imadoueddine, Philippe Thierry, Hervé Chauris, and Leonardo Borges (2011). "Tips and tricks for finite difference and i/o-less FWI". In: *SEG International and Exposition 81st Annual Meeting, San Antonio*. San Antonio, USA, pp. 3174–3178 (see p. 34).

Jin, Guohua and J Mellor-Crummey (2005). "Using space-filling curves for computation reordering". In: *Proceedings of the Los Alamos Computer Science Institute Sixth Annual Sympo- sium*, pp. 1–9 (see p. 47).

Keckler, S.W., W.J. Dally, B. Khailany, M. Garland, and D. Glasco (2011). "GPUs and the Future of Parallel Computing". In: *Micro, IEEE* 31.5, pp. 7–17 (see p. 8).

Kelly, K., R. Ward, S. Treitel, and R. Alford (1976). "Synthetic Seismograms: A Finite-Difference Approach". In: *GEOPHYSICS* 41.1, pp. 2–27 (see p. 28).

Khabou, Amal (2013). "Dense matrix computations: communication cost and numerical stability." PhD thesis. Université Paris Sud-Paris XI (see p. 11).

Kogge, P.M. and T.J. Dysart (2011). "Using the TOP500 to trace and project technology and architecture trends". In: *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pp. 1–11 (see p. 8).

Kogge, Peter, Study Lead, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R Stanley Williams, and Katherine Yelick (2008). *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems*. Tech. rep. (see p. 1, 3, 7, 8).

Kosloff, D. and E. Baysal (1982). "Forward modeling by a Fourier method". In: *GEOPHYSICS* 47.10, pp. 1402–1412 (see p. 27).

Krueger, Jens, David Donofrio, John Shalf, Marghoob Mohiyuddin, Samuel Williams, Leonid Oliker, and F.J. Pfreundt (2011). "Hardware/Software Co-design for Energy-Efficient Seismic Modeling". In: *Proceedings of SC2011* (see p. 9, 12, 14).

Lee, Jaekyu, Hyesoon Kim, and Richard W. Vuduc (2012). "When Prefetching Works, When It Doesn't, and Why". In: *TACO* 9.1, p. 2 (see p. 42).

Lipshitz, Benjamin, Grey Ballard, Oded Schwartz, and James Demmel (Nov. 2012). "Communication-Avoiding Parallel Strassen: Implementation and Performance". In: *SC'12 CD-ROM: Conference on High Performance Computing Networking, Storage and Analysis*. Salt Lake City, UT, USA: ACM SIGARCH/IEEE Computer Society (see p. 11).

Liu, Wei, Tomas Nemeth, Alexander Loddoch, Joseph Stefani, Ray Ergas, Ling Zhuo, Bill Volz, Oliver Pell, and James Huggett (2009). "Anisotropic reverse time migration using coprocessors". In: *SEG Technical Program Expanded Abstracts 2009*. Chap. 610, pp. 3040–3044 (see p. 83).

Loh, G.H. (2008). "3D-Stacked Memory Architectures for Multi-core Processors". In: *Computer Architecture, 2008. ISCA '08. 35th International Symposium on*, pp. 453–464 (see p. 5).

Marin, Gabriel and John Mellor-Crummey (2004). "Cross-architecture Performance Predictions for Scientific Applications Using Parameterized Models". In: *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS '04/Performance '04. New York, NY, USA: ACM, pp. 2–13 (see p. 59).

Mattson, R. L., J. Gecsei, D. R. Slutz, and I. L. Traiger (June 1970). "Evaluation Techniques for Storage Hierarchies". In: *IBM Syst. J.* 9.2, pp. 78–117 (see p. 58).

McCalpin, John D (2000). *STREAM: Sustainable memory bandwidth in high performance computers* (see p. 40).

Mellor-Crummey, John, David Whalley, and Ken Kennedy (June 2001). "Improving Memory Hierarchy Performance for Irregular Applications Using Data and Computation Reorderings". In: *International Journal of Parallel Programming* 29.3, pp. 217–247 (see p. 47).

Micikevicius, Paulius (Mar. 2009). "3D finite difference computation on GPUs using CUDA". In: *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, (2nd GPGPU'09) in conjunctions with (14th ASPLOS'09)*. Vol. 383. ACM International Conference Proceeding Series. Washington, DC, USA: ACM, pp. 79–84 (see p. 83).

Moczo, Peter, Johan OA Robertsson, and Leo Eisner (2007). "The finite-difference time-domain method for modeling of seismic wave propagation". In: *Advances in Geophysics* 48, pp. 421–516 (see p. 28).

Nguyen, Anthony, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey (2010). "3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '10. Washington, DC, USA: IEEE Computer Society, pp. 1–13 (see p. 44, 52).

Oliveira Castro, Pablo de, Eric Petit, Asma Farjallah, and William Jalby (2013). "Adaptive sampling for performance characterization of application kernels". In: vol. 25. 17, pp. 2345–2362 (see p. 48).

Operto, S, Jean Virieux, and A Ribodetti (2009). "Finite-difference frequency-domain modeling of viscoacoustic wave propagation in 2D tilted transversely isotropic (TTI) media". In: *Geophysics* 74.5 (see p. 28).

Ortigosa, Francisco, Mauricio Araya Polo, Felix Rubio, Jose Maria Cela, Raud de la Cruz, and Mauricio Hanzich (2008). "Evaluation of 3d rtm on hpc platforms". In: *2008 SEG Annual Meeting* (see p. 33, 83).

Pascucci, Valerio and Randall J. Frank (2001). "Global static indexing for real-time exploration of very large regular grids". In: *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*. Supercomputing '01. Denver, Colorado: ACM, pp. 2–2 (see p. 47).

Perrone, M., Lurng-Kuo Liu, Ligang Lu, K. Magerlein, Changhoan Kim, I. Fedulova, and A. Semenikhin (2012). "Reducing Data Movement Costs: Scalable Seismic Imaging on Blue Gene". In: *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 320–329 (see p. 82).

Pratt, R Gerhard (1999). "Seismic waveform inversion in the frequency domain, Part 1: Theory and verification in a physical scale model". In: *Geophysics* 64.3, pp. 888–901 (see p. 28).

Raman, Karthik (2013). *Optimizing Memory Bandwidth on Stream Triad*. https://software.intel.com/en-us/articles/optimizing-memory-

bandwidth-on-stream-triad. [Online; accessed 09-September-2014] (see p. 70).

Rivera, Gabriel and Chau-Wen Tseng (2000). "Tiling optimizations for 3D scientific computations". In: *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*. Supercomputing '00. Washington, DC, USA: IEEE Computer Society (see p. 43, 44, 52).

Sercel (2014). *What on Earth is Geophysics?* [Online; accessed 06-January-2014]. URL: http://www.sercel.com/about/what-is-geophysics.aspx (see p. 18).

Shalf, John, David Donofrio, Curtis Janssen, and Dan Quinlan (2011a). *CoDEx Web page*. URL: http://www.nersc.gov/research-and-development/exascale-computing/codex-project/ (see p. 12).

Shalf, John, Sudip Dosanjh, and John Morrison (2011b). "Exascale computing technology challenges". In: *Proceedings of the 9th international conference on High performance computing for computational science*. VECPAR'10. Berkeley, CA: Springer-Verlag, pp. 1–25 (see p. 3).

Shalf, John, Dan Quinlan, and Curtis Janssen (2011c). "Rethinking Hardware-Software Codesign for Exascale Systems". In: *IEEE Computer* 44.11, pp. 22–30 (see p. 6, 12, 13).

Shen, Xipeng, Yutao Zhong, and Chen Ding (2003). "Regression-based multi-model prediction of data reuse signature". In: (see p. 59).

Sheriff, R.E. and L.P. Geldart (1995). *Exploration Seismology*. Cambridge University Press (see p. 15, 18).

Stevens, Rick, Andrew White, Sudip Dosanjh, Al Geist, Brent Gorda, Kathy Yelick, John Morrison, Horst Simon, John Shalf, Jeff Nichols, and Mark Seager (2009). *Scientific Grand Challenges Architectures and technology for extreme scale computing*. Tech. rep. (see p. 6).

Symes, William W. (2007). "Reverse time migration with optimal checkpointing". In: *Geophysics* 72.5, SM213–SM221 (see p. 86).

Treibig, Jan, Georg Hager, and Gerhard Wellein (2010). "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments". In: *CoRR* abs/1004.4431 (see p. 56).

Treibig, Jan, Gerhard Wellein, and Georg Hager (2011). "Efficient multicore-aware parallelization strategies for iterative stencil computations". In: *Journal of Computational Science* 2.2, pp. 130 –137 (see p. 46).

Valsalam, Vinod and Anthony Skjellum (2002). "A framework for high-performance matrix multiplication based on hierarchical abstractions, algorithms and opti-

mized low-level kernels". In: *Concurrency and Computation: Practice and Experience* 14.10, pp. 805–839 (see p. 48).

Versteeg, R. (1993). "Sensitivity of prestack depth migration to the velocity model". In: *GEOPHYSICS* 58.6, pp. 873–882 (see p. 83).

Virieux, J and S Operto (2009). "An overview of full-waveform inversion in exploration geophysics". In: *Geophysics* 74.6, WCC1 (see p. 28).

Virieux, Jean, Henri Calandra, and R.É. Plessix (2011). "A review of the spectral, pseudo-spectral, finite-difference and finite-element modelling techniques for geophysical imaging". In: *Geophysical Prospecting* 59.5, pp. 794–813 (see p. 21, 23, 28).

Wellein, G., G. Hager, T. Zeiser, M. Wittmann, and H. Fehske (2009). "Efficient Temporal Blocking for Stencil Computations by Multicore-Aware Wavefront Parallelization". In: *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*. Vol. 1, pp. 579–586 (see p. 46).

Williams, Samuel, Andrew Waterman, and David Patterson (Apr. 2009). "Roofline: an insightful visual performance model for multicore architectures". In: *Commun. ACM* 52.4, pp. 65–76 (see p. 35).

Wittmann, Markus, Georg Hager, Jan Treibig, and Gerhard Wellein (2010). "Leveraging Shared Caches For Parallel Temporal Blocking Of Stencil Codes On Multicore Processors and Clusters". In: *IPDPS* (see p. 46).

Woo, Dong Hyuk, Nak Hee Seong, D.L. Lewis, and H.-H.S. Lee (2010). "An optimized 3D-stacked memory architecture by exploiting excessive, high-density TSV bandwidth". In: *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pp. 1–12 (see p. 5).

Zhong, Y., S.G. Dropsho, Xipeng Shen, A Studer, and Chen Ding (2007). "Miss Rate Prediction Across Program Inputs and Cache Configurations". In: *Computers, IEEE Transactions on* 56.3, pp. 328–343 (see p. 59).

List of Figures

List of Algorithms

List of Tables